Army Research Laboratory

Database Entity Persistence with Hibernate for the Network Connectivity Analysis Model

by Andrej Bevec

ARL-TR-6893 April 2014

NOTICES

Disclaimers

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5068

ARL-TR-6893 **Apri 2014**

Database Entity Persistence with Hibernate for the Network Connectivity Analysis Model

Andrej Bevec
Survivability/Lethality Analysis Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* April 2014	**2. REPORT TYPE** Final		**3. DATES COVERED (From - To)** 27 January 2009–22 June 2013
4. TITLE AND SUBTITLE Database Entity Persistence with Hibernate for the Network Connectivity Analysis Model			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Andrej Bevec			**5d. PROJECT NUMBER**
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-SLB-G Aberdeen Proving Ground, MD 21005-5068			**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-6893
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			
13. SUPPLEMENTARY NOTES			
14. ABSTRACT This report addresses the design and architecture of data persistence for the Network Connectivity Analysis Model application developed in the Java language and using the Hibernate Application Programming Interface as the object-relational mapping library. The report also addresses the database architecture, the Class/Entity domain model design, and the Java design patterns incorporated, such as the Factory and Data Access Object design patterns for the Hibernate implementation.			
15. SUBJECT TERMS hibernate, persistence, entity persistence, entity, NCAM, network connectivity analysis model, database architecture			

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Andrej Bevec
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** (*Include area code*)
Unclassified	Unclassified	Unclassified	UU	94	410-395-0291

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Preface

A main objective of any software development application is the ability to save data that may be generated by the run-time environment and saved into a database or multiple databases as the program generates data and needs to perform create, read, update, and delete database operations.

Presented in this report is the design of data persistence, or saving of data generated by the application, of the Network Connectivity Analysis Model application developed in the Java language and using the Hibernate Application Programming Interface as the object-relational mapping library. The report addresses the database architecture, the Class/Entity domain model design, and the Java design patterns incorporated, such as the Factory and Data Access Object design patterns for the Hibernate implementation.

1. Background

The Network Connectivity Analysis Model (NCAM) software models the physical layer of the Open Systems Interconnection model of wireless communication for the U.S. Army's platforms by addressing the wireless link qualities between stationary or moving platforms. The platforms may be traversing benign or hostile electronic warfare environments.

The NCAM software is divided into two main Java projects, NCAM and NCAMLib. The NCAM project includes the classes associated with the graphical user interface while the NCAMLib project is used as a library by NCAM. NCAMLib is where all of the modeling and simulation takes place. This separation provides for the model-view-controller type of software architecture.

The NCAM design is broken up into object-oriented (OO) modules programmed in the Java language. The modules, i.e., Deployment, Propagation, Antenna, Noise, Link Budget, and Connectivity, comprise the NCAM software. Each specific module tackles a specific physics or a physical phenomenon problem to determine the overall link quality among the platforms specified for a NCAM run.

Java, Netbeans, Hibernate, and the MySQL database management system are the pieces of the software development environment that are used to develop the data persistence architecture for NCAM/NCAMLib.

Hibernate is a high-performance object/relational persistence and query service that takes care of the mapping from Java classes to database tables and from Java data types to structured query language (SQL) data types. It provides data query and retrieval facilities that significantly reduce development time. Hibernate's design goal is to relieve the developer from the majority of common data persistence-related programming tasks by eliminating the need for manual, handcrafted data processing using the SQL and Java Database Connectivity (JDBC). However, unlike many other persistence solutions, Hibernate does not hide the power of SQL from the user and guarantees that that investment in relational technology and knowledge is as valid as always. Working with both OO software and relational databases can be cumbersome and time consuming. Development costs are significantly higher due to a paradigm mismatch between how data is represented in objects versus relational databases. Hibernate can significantly reduce development time, and as a provider of object/relational persistence solution it will significantly reduce lines of code, provide a buffer between the two data representations, and enable a more elegant use of OO on the Java side—all while keeping the relational schema normalized and guaranteeing data integrity. Many software developers and architects estimate that up to 30% of their code is needed to deal with this infrastructure concern. Hibernate directly addresses this challenge by providing the ability to map an object model's data representation to a relational data model and its corresponding database schema (Hibernate, 2013).

Hibernate generates the SQL calls through the JDBC application programming interface (API) and relieves the Java software programmer from using result-set handling and object casting, which is the norm for non-Hibernate implementations. Hibernate keeps the application portable to all supported SQL relational database management systems. From our experience in profiling the run time of NCAM in Netbeans, there was little overhead for Hibernate for reads and writes to the database but significant overhead for bulk deletes. This was fixed by providing SQL query injection with Hibernate for bulk deletes.

Hibernate provides transparent persistence for entities with the only major requirement that the Java class has a zero argument constructor. Hibernate provides a checking feature that avoids unnecessary database write actions by performing SQL updates only on the modified fields of persistent entities.

In transforming Entity objects, or entities, into corresponding table entries in a database, Hibernate requires metadata that transforms data from one representation to the other using extensible markup language (XML) files or annotations in the Java source class annotated as Entity. Annotations are identified by the "@" symbol in the source code; e.g., a class to be persisted would be annotated with "@Entity" at the top of the class. These mapping files or annotations provide the information required by Hibernate to properly map a class to a database table, as well as how to handle the SQL queries to persist the entities. In this project, Hibernate annotations were used instead of XML transform files. This is a cleaner, more compact, and less time-consuming programming approach.

2. Database Architecture

NCAM is a database-intense application because of the amount of data it generates during the run time. It interacts with three databases: (1) the "scenario," which is a read/write database for the end user's scenario data and the output generated during the run time, (2) the read-only "blueSystems" database, which holds information on friendly force platforms and common scenario information such as antenna patterns, and (3) the read-only "redSystems" database, which holds information on enemy platforms. The NCAM database architecture, shown in figure 1, shows how NCAM interacts via the Hibernate layer and JDBC to interact with the three databases.

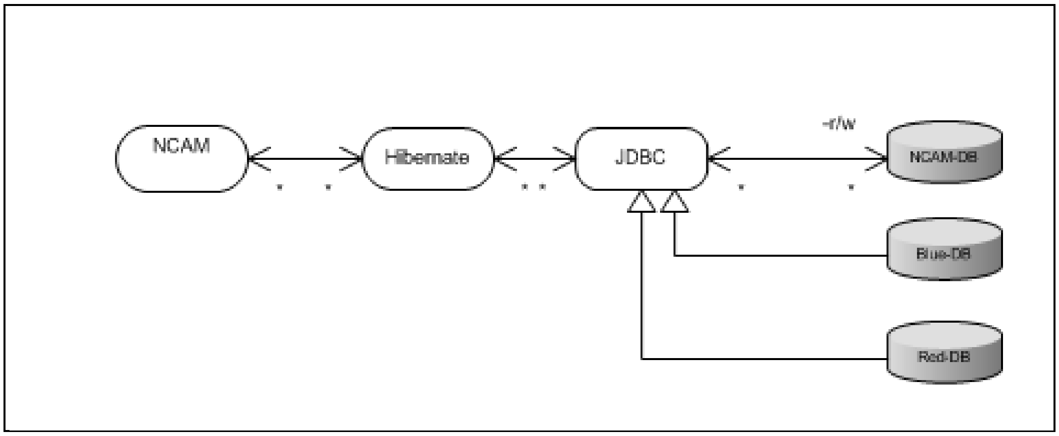


Figure 1. NCAM database architecture.

The Hibernate API provides a static method, “Hibernate.recreateDatabase,” for recreating the database schema when the entity domain model changes. A simple program was written to take advantage of this capability (Project NCAMLib persist.CreateDBSchema.java). This program is part of the NCAM source code to recreate the database schema when the NCAM domain entities were revised. The database schema for each of the three databases is created via Hibernate using the Entity domain model described in section 9 of this report. This saves a significant amount of time in the software development effort because the domain Entity class structure is mapped to the relational database schema by Hibernate. The data dictionary for the “scenario” database showing the mapping of the NCAM entity domain model to the database tables is shown in appendix B.

3. NCAM Class/Entity Model

An instantiated class is an object. An object that is annotated as an Entity that implements the Java Serializable interface has a zero argument constructor, conforms to public getters and setters for its private attributes, and is persisted as an Entity object. In this report, an Entity refers to the class annotated as an Entity or may be thought of as an Entity set. An Entity object refers to the instantiated class itself. All Entity diagrams in this report thus will show Entity relationship sets or class relationships among each other.

A Java class that is annotated as an Entity identifies to Hibernate that the instantiated class, i.e., object, may be persisted to a relational database. The Entity model, as it pertains to NCAM, may be thought of as a subset of all the classes that, when instantiated, may be persisted to a database. Persisting an object means saving that object to a database for future use by a program using the Hibernate Object-Relational Mapping (ORM) framework. When instantiated, a Java class annotated as an Entity may be persisted via Hibernate into the database for future Create, Read, Update, or Delete (CRUD) operations.

How these classes and entities in NCAM relate to one another is shown in a conceptual class/entity domain design, shown in figure 2, which also shows classes that are only part of the NCAM class domain model, i.e., classes that do not need to be persisted but are included here for a more complete understanding of the Entity relationships with the NCAM domain class structure. NCAM-only classes are highlighted in gray; Entity-annotated classes are shown in white. To keep the figures simple, attributes and relationships for the entities and classes are omitted. The cardinality of the entities is shown with an asterisk (*) representing a “many” relationship and the numeral 1 representing a relationship of one.

When persisted, Entity objects are Java objects that are saved into a database as records that may be retrieved later and used as necessary. An Entity object is a row in a database table. The table itself is the mapped Entity set, which corresponds to the class annotated as an Entity.

Mapping Java classes to database tables is accomplished via the configuration of an XML file or by using Java Annotations. In this project, annotations were used throughout the project except for configuring the initial Hibernate startup files, i.e., hibernate.cfg.xml.

3.1 Cardinality of the NCAM Entity Model

Cardinality refers to the number of instances of an entity, and deals with relationships between entities (figures 2–5). Relationships between entities may be one-to-one, one-to-many, many-to-one, and many-to-many, identified as 1:1, 1:*, *:1, or *.* in the figures. The most common, for example, a one-to-many relationship, is shown between the AntennaModel and AntennaPattern classes. An instance of AntennaModel is an antenna that may have more than one antenna pattern associated with it, e.g., if an antenna is driven at a different frequency or is located vertically or horizontally on a platform, the antenna will produce different antenna patterns for one unique physical antenna. Thus, a one-to-many (1:*) cardinality exists between AntennaModel and the AntennaPattern entities (figures 2–5).

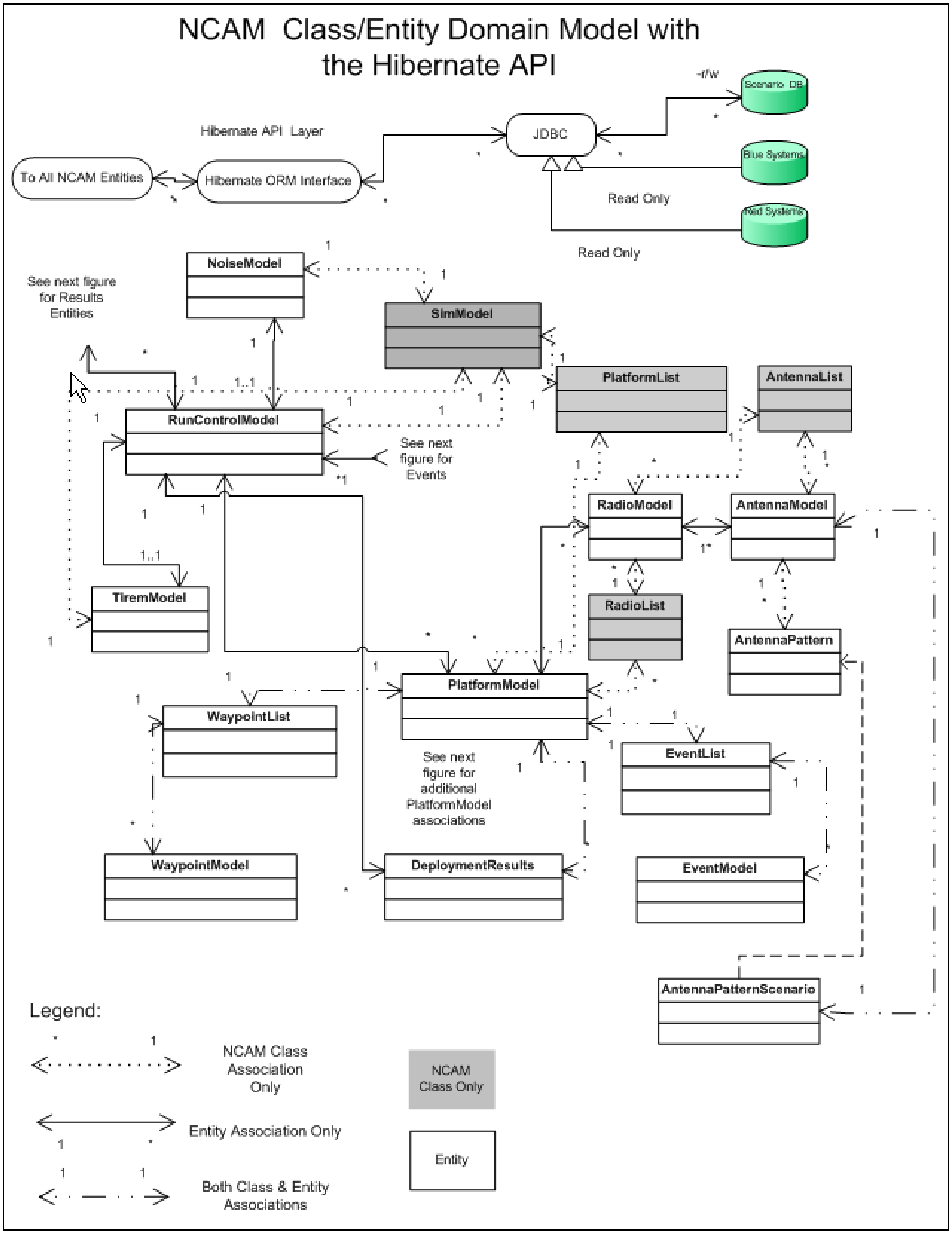


Figure 2. NCAM class/entity model.

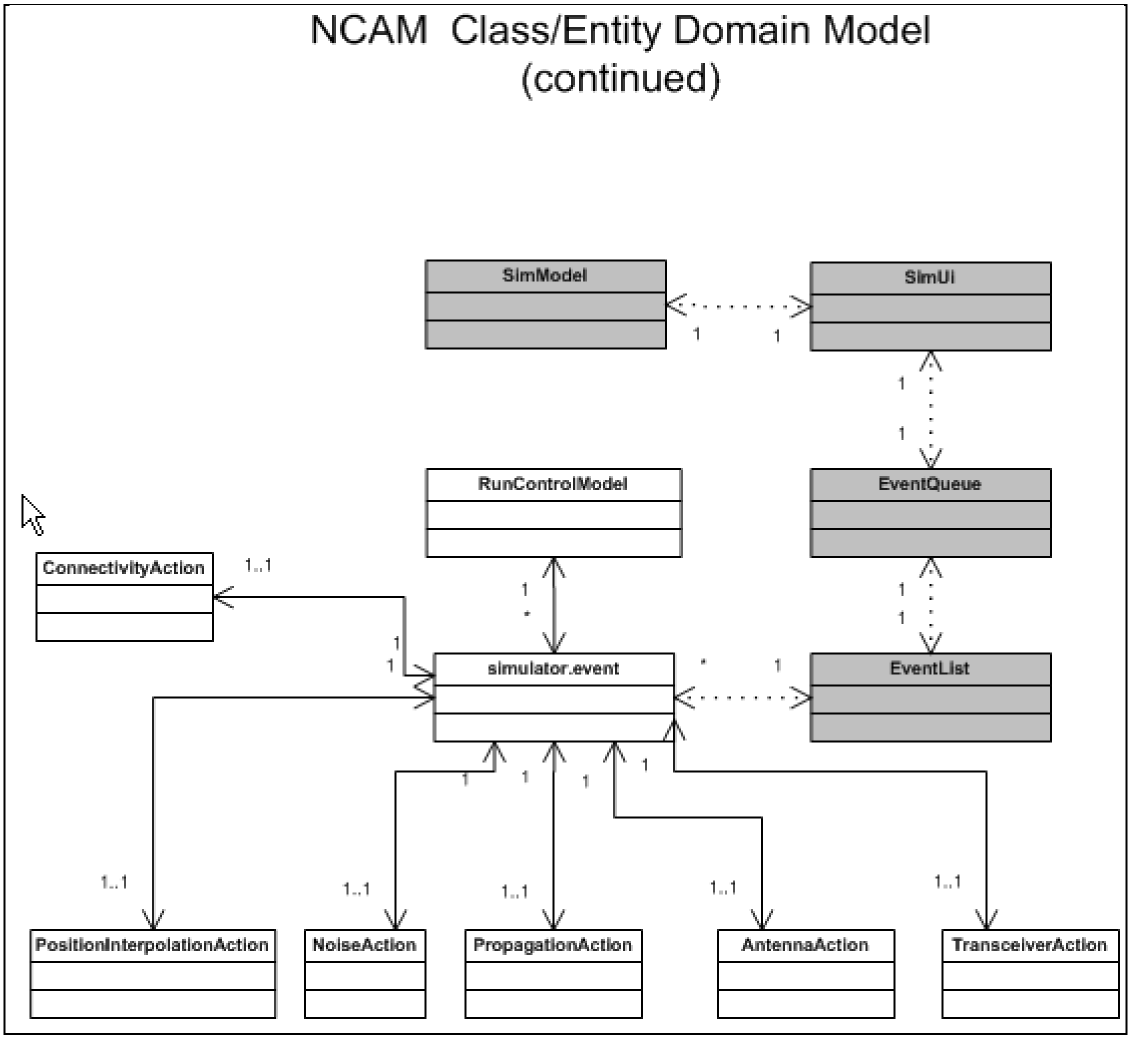


Figure 3. NCAM class/entity model for associations for "simulator.event."

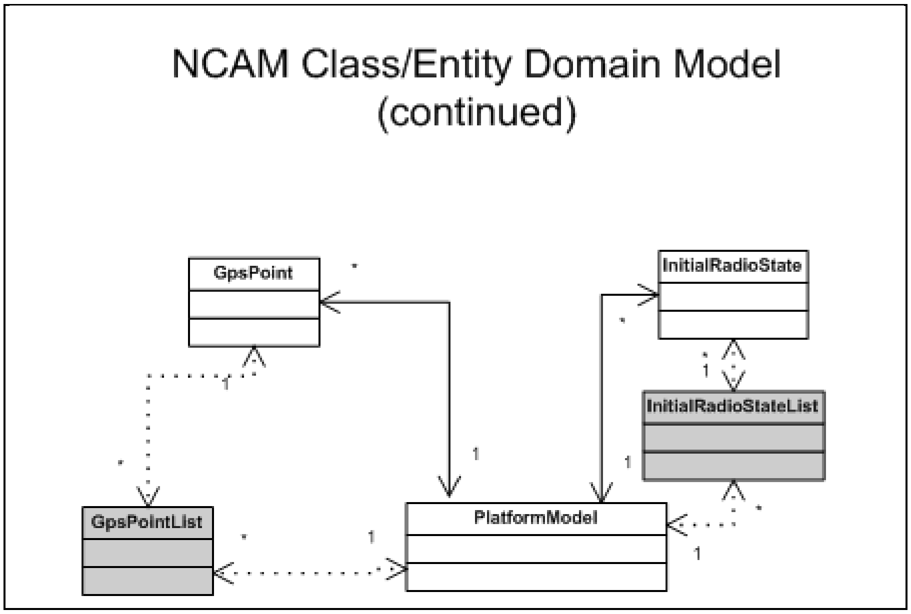


Figure 4. NCAM class/domain model associations for "PlatformModel."

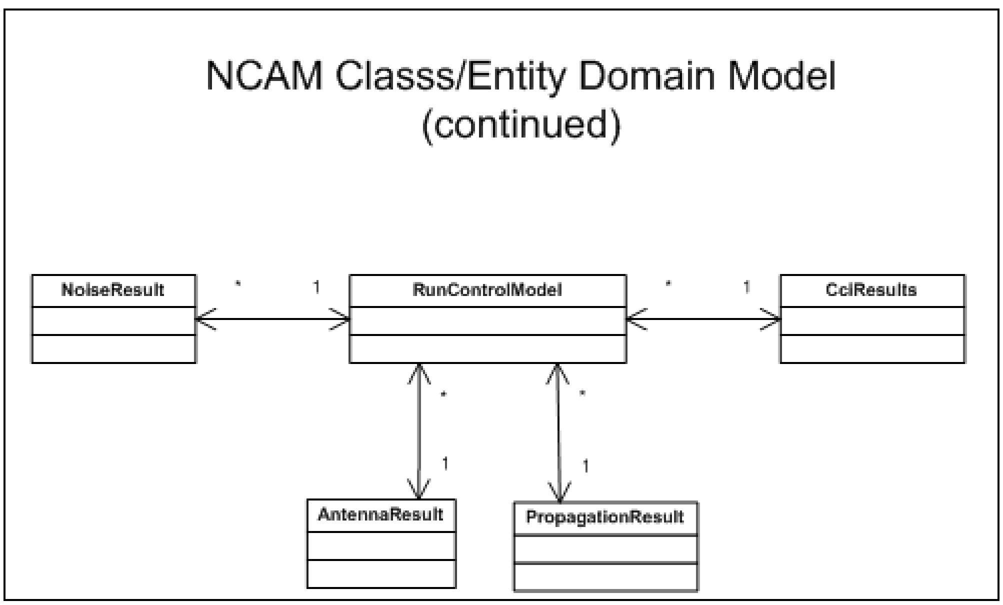


Figure 5. Result entity associations for "RunControlModel."

4. Data Access Object – Design Pattern

The data acess object (DAO) is an important design pattern because it provides an abstract interface that hides the underlying database implementation and the Hibernate framework.

Hibernate is a very powerful and elegant way to implement database interactions, but Java programmers wish to deal with classes and objects and not worry about understanding SQL database syntax or performing CRUD operations to understand SQL. The DAO is a Java design pattern implemented in NCAM that hides the implementation of the Hibernate persistence framework from the Java programmer, making it easier to implement future changes to the database layer. In the DAO design pattern, the NCAM application is provided with Java interfaces for accessing data; the actual implementation of those interfaces are performed by classes that may be replaced if the need arises. Providing Java programmers DAOs instead of coding directly to the Hibernate API simplifies application development, insulates changes that may occur in the database from other layers in the application, and eases interaction with the databaseand overall code maintainability.

One of the goals in designing software is to factor out common or reusable methods. Thus, a Hibernate implementation of the DAO pattern factors out CRUD methods into reusable, abstract, and inheritable parent classes.

4.1 GenericDAO

For NCAM, a GenericDAO—Java 5 Generics Interface—is created that hides both the underlying database implementation and the mechanism or the framework being used to persist data to the database (McKenzie, 2008, p. 388). GenericDAO defines the most important methods and functionality that our application needs, namely the CRUD from our persistence layer. The GenericDAO provides the abstraction necessary if the underlying database implementation is changed and for easier Java coding by hiding the underlying complexity of the Hibernate persistence implementation. The most general interface in NCAM is the GenericDAO found in the persist package of the NCAM project (Markowski et al., 2012), as follows:

```
interface GenericDAO<T, ID extends Serializable> {

 T finDecibelsyPrimaryKey(ID id);

List<T> finDecibelsyExample(T exampleInstance, String[] excludeProperty);

List<T> findAll(int startIndex, int fetchSize);
List<T> findAll();
List<T> findAll();
List<T> finDecibelsyExample(T exampleInstance);
List<T> findKidsOfParent(T exampleInstance, String parent, Long parentPrimaryKey);
List<T> findKidsOfParentMaxResultsOne(T exampleInstance, String parent, Long parentPrimaryKey);
```

```
List<T> findKidsOfParentByTime(T exampleInstance, String parent, Long parentPrimaryKey, double time);
T saveOrUpdate(T entity);
T merge(T entity);
void delete(T entity); void beginTransaction(); void commitTransaction();
.
```

etc.,

. }

These abstract methods defined in the interface are implemented in a Hibernate DAO abstract class, i.e., HibernateDAO. The interface signature provides for querying of the database for objects of generic type T, with a parameter ID, where parameter *id* is the primary key of the object and a returned object of type T from the database. The parameter <T> represents the class type the DAO manages, and parameter <ID> is the ID type of a serializable object. With Java Generics, no type casting is required for the returned object.

The abstract GenericDAO methods are identified by the following categories:

- Finder Methods
    - T finDecibelsyPrimaryKey(ID id)
    - List <T> findAll(int startIndex, int fetchSize)
    - List<T> finDecibelsyExample(T exampleInstance, String[] excludeProperty)
- Persistence Method
    - T saveOrUpdate(T entity);
- Delete Method
    - void delete(T entity);
- Transactional Methods
    - void beginTransaction();
    - void commitTransaction();

4.2 Child DAO Interfaces

For subclassed DAO interfaces, namely interfaces for the entities that we wish to persist, one does not need to define many new methods since the ones inherited will be included from the parent, i.e., GenericDAO. If new business logic is required for a specific DAO, that logic needs to be included in that child interface and not the GenericDAO. As a general design rule, one creates a DAO for each class defined in the problem domain, which is an Entity that needs to be persisted. These DAOs are the only ones that will be needed by the Java programmer because the Hibernate implementation of the entity-specific DAOs will not be seen by the Java programmers. Therefore, it is possible to replace the Hibernate implementation with any other future technology or ORM framework if the need arises. The child DAOs concretely define the generic T type and generic Serialable ID type, which is part of the GenericDAO interface, as shown in the following with the AntennaModelDAO:

```
public interface AntennaModelDAO extends GenericDAO<AntennaModel, Long>{

}
```

As can be seen, the T type is defined as *AntennaModel* and the ID type as *Long.*

5. HibernateDAO

The concrete implementation of the GenericDAO occurs in the HibernateDAO abstract class. With comments removed, the abstract class that follows (Markowski et al., 2012) shows some of the implementation. For the complete implementation, refer to the source code in the NCAMLib project under the “dao” package:

```
public abstract class HibernateDAO<T, ID extends Serializable> implements
      GenericDAO<T, ID> {

   private Class<T> persistentClass;

   public HibernateDAO(Class c) {
      persistentClass = c;
   }

   public  T findByPrimaryKey (ID id) {
      return (T) HibernateUtil.getSession().load(persistentClass, id);

   }

   public List<T>  findByExample (T exampleInstance, String[] excludeProperty) {
      Criteria crit = HibernateUtil.getSession().createCriteria(persistentClass);
      Example example = Example.create(exampleInstance);
      if (excludeProperty != null) {
         for (int i = 0; i < excludeProperty.length; i++) {
            example.excludeProperty(excludeProperty[i]);
         }
      }
      crit.add(example);
      return crit.list();
```

```
    }

    public List<T>  findByExample (T exampleInstance) {
        Criteria crit = HibernateUtil.getSession().createCriteria(persistentClass);
        Example example = Example.create(exampleInstance);
        crit.add(example);
        return crit.list();

    public List<T> findKidsOfParent(T exampleInstance, String parent, Long pk) {
        List results =
HibernateUtil.getSession().createCriteria(persistentClass).createCriteria(parent).add(Restrictio
ns.eq("primaryKey", pk)).list();

        return results;
    }

  public List<T> findKidsOfParentMaxResultsOne(T exampleInstance, String parent, Long pk) {
        //need to chain criteria objects to get proper results as below
        Criteria criteria =
HibernateUtil.getSession().createCriteria(persistentClass).createCriteria(parent).setMaxResults
(1).add(Restrictions.eq("primaryKey", pk));
        return criteria.list();
    }

.. etc
}
```

---

6. Hibernate Criteria API

Hibernate provides for various methods for querying Hibernate-persisted databases. It allows one to express queries using standard SQL, Hibernate Query Language (HQL), the Criteria API, or a combination of all. HQL is an OO version of SQL but much less verbose. If the Java/Hibernate database designer wants no SQL syntax introduced into the Java code, Hibernate provides the Criteria API. The Criteria API provides for Query by Criteria (QBC) and Query By Example (QBE) OO queries. In NCAM, the QBC and QBE APIs are used extensively, as they are with HQL where database speed is necessary. These APIs provide an elegant OO solution to querying the database on the fly. An example of the Criteria API being leveraged is shown in the HibernateDAO abstract class, findbyExample *{}* method.

7. Concrete HibernateDAO classes

With the abstract HibernateDAO class providing most of the implementations for the GenericDAO and creating the corresponding concrete subclasses for the entities, persistence is accomplished by extending the HibernateDAO abstract class, implementing the corresponding DAO interface, and providing concrete class names for the generic types defined by the GenericDAO interface.

The example of the HibernateAntennaModelDAO follows (Markowski et al., 2012):

```
public class HibernateAntennaModelDAO extends HibernateDAO<AntennaModel, Long>
implements AntennaModelDAO {

   public HibernateAntennaModelDAO() {

      super(AntennaModel.class);
   }
}
```

The concrete HibernateAntennaModelDAO class extends the HibernateDAO java abstract class passing in the AntennaModel and Long objects via the Generics description. It calls its parent class constructor via the super keyword.

7.1 DAO Hibernate Design in NCAM

A class and interface DAO diagram for the NCAM DAO design implementation is shown in figure 6. Note that (1) the GenericDAO interface is implemented by the HibernateDAO abstract class, (2) the individual DAO interfaces extend the GenericDAO, and (3), in turn, the concrete Hibernate classes implement each corresponding interface and extend the Hibernate DAO. With this design pattern in place, a very abstract interface is provided that hides both the underlying database implementation and the framework, also providing OO Java coding using DAOs. Factory, another design pattern incorporated into NCAM and presented in section 8, will completely hide the Hibernate layer from the Java programmer.

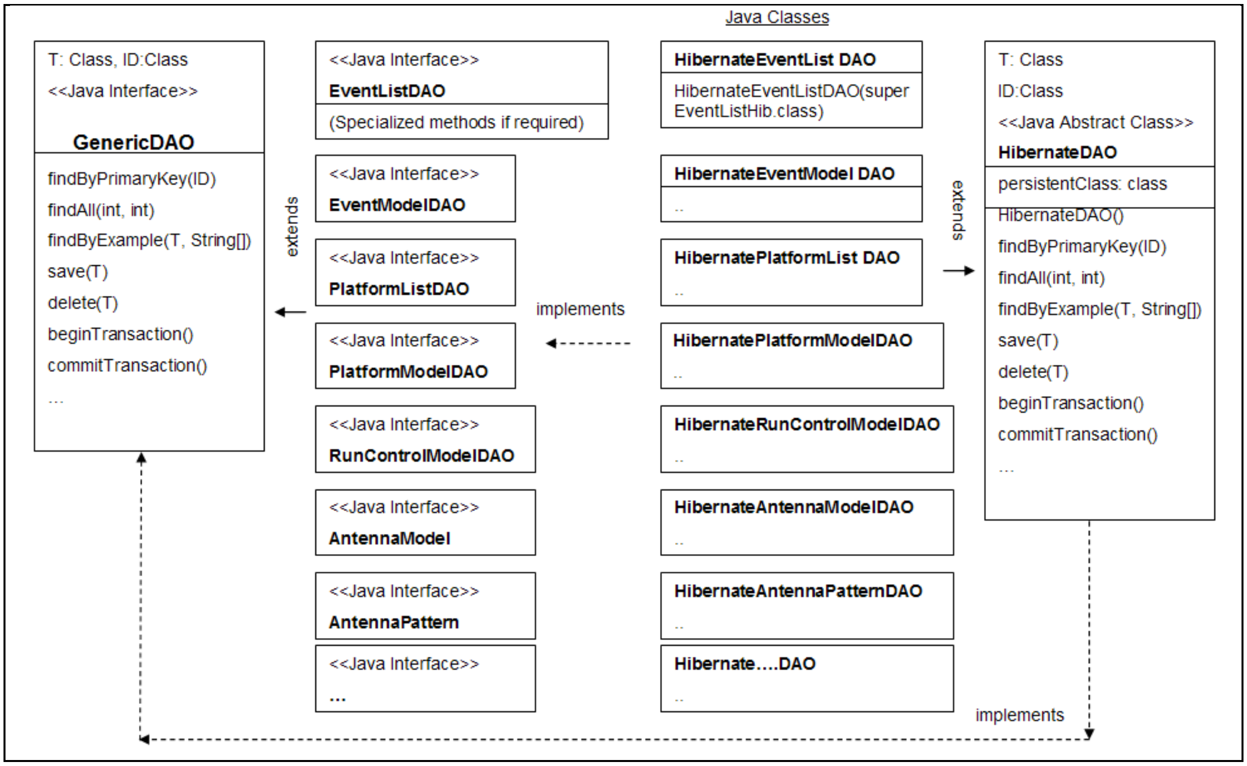


Figure 6. DAO design for hibernate CRUD interactions with databases.

8. Factory Design Pattern Incorporation

A Factory pattern helps to model an interface for creating an object that at creation time can let its subclasses decide which class to instantiate. For example, if a Java programmer wanted to instantiate the AntennaModelDAO to do some database CRUD operations, one would have to do something like AntennaModelDAO antennaModelDAO = new HibernateAntennaModel( ).

Obviously, that code snippet does not hide the back-end implementation of Hibernate if the word “Hibernate” can be seen and appearing throughout the code. To make sure the Java programmer never sees any of the concrete Hibernate classes, a Factory design pattern is used. Any good DAO design pattern implements the Factory design pattern (Gamma et al., 1994).

By implementing the Factory design pattern, the Java programmer will be given an abstract class called DAOFactory, which contains abstract methods for accessing each of the DAO classes of interest, e.g., PlatformModelDAO, RunControlModelDAO, and AntennaModelDAO.

The abstract DAOFactory class will have a single static invocable method that will return an instantiated instance of the DAOFactory (McKenzie et al., 2008, p 399). The concrete class that implements the DAOFactory is HibernateDAOFactory, which is referenced inside the

DAOFactory via the constant variable, FACTORY_CLASS. A partial listing of this abstract class without code comments follows (Markowski et al., 2012):

```
public abstract class DAOFactory {

public static final Class FACTORY_CLASS = HibernateDAOFactory.class;

   public static DAOFactory getFactory() {
      try {
         return (DAOFactory) FACTORY_CLASS.newInstance();

      } catch (Exception e) {
         throw new RuntimeException("Could not create Factory");
      }

 public abstract EventListDAO getEventListDAO();

 public abstract EventModelDAO getEventModelDAO();

public abstract AntennaModelDAO getAntennaModelDAO();
… etc

   }
```

DAOFactory is the class the Java programmers use to gain access to DAO objects to persist their Java entities to the database. For example, to gain access to the AntennaModelDAO,

```
DAOFactory factory = DAOFactory.getFactory();
AntennaModelDAO antennaModelDAO = factory.getAntennaModelDAO;
```

Thus, there are no references that the underlying persistence layer is implemented via Hibernate. What is gained here is that the underlying implementation may be changed from Hibernate to JDBC or Java Data Objects (JDOs) API as long as the same DAO interfaces, such as AntennaModelDAO, PlatformModelDAO, etc., are implemented. So we gain complete flexibility on the data side as to how the persistence layer is managed and complete persistence-layer independence on the client side of the application. The DAO pattern with the Factory pattern demonstrates the separation of concerns for the NCAM software application.

A very important aspect of the design is how the static getFactory() method of the DAOFactory is implemented. The getFactory method returns an instance of a class that implements the abstract methods defined in the DAOFactory class. The class type is coded as a static final class variable in the DAOFactory, instantiated and returned from the getFactory method. Thus, the implementation of DAOFactory as HibernateDAOFactory is all hidden from the Java programmer and may be changed in the future if the need arises to something like JDODAOFactory or JDBCDAOFactory class, etc.

The concrete class that implements the DAOFactory is named HibernateDAOFactory, shown in the following with a partial listing:

```
public class HibernateDAOFactory extends DAOFactory {

public EventListDAO getEventListDAO() {
      return new HibernateEventListDAO();
   }

  public EventModelDAO getEventModelDAO() {
      return new HibernateEventModelDAO();
   }
..
  public AntennaModelDAO getAntennaModelDAO() {
      return new HibernateAntennaModelDAO();
   }
..
}
```

The Factory design pattern implementation for DAO is shown in figure 7, which shows how DAOs are accessed through the DAOFactory.

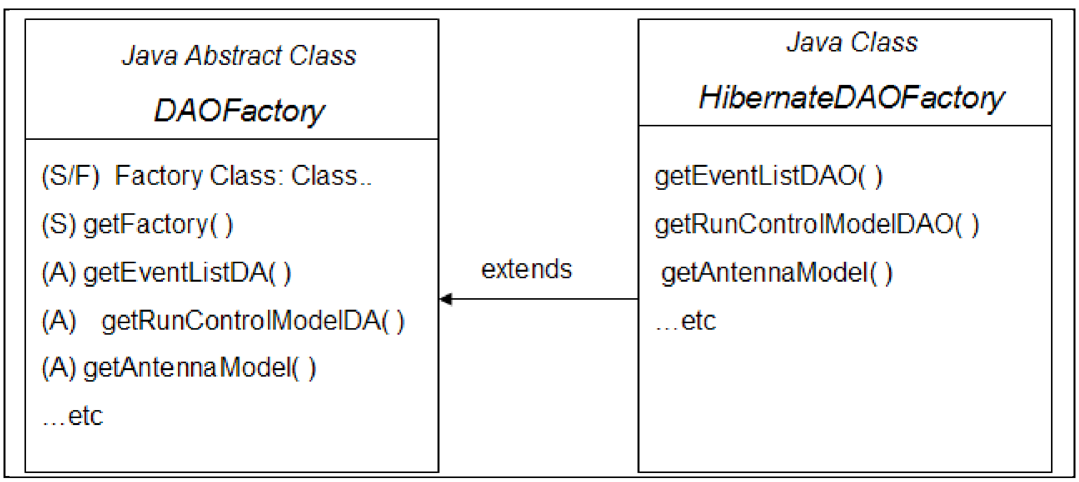


Figure 7. Factory pattern for DAOs, where S/F = static and final and A = abstract.

9. An NCAM Entity Persistence – Java Coding Example

This section presents a Java coding example of persisting AntennaModel Entity objects and a discussion of the concepts involved.

9.1 Hibernate Annotations

This is not a primer on Hibernate, so the reader may need to consult the respective documentation. Older versions of Hibernate use the XML mapping files for Hibernate to transform everything properly into relational SQL database structures. Hibernate Annotations may be used instead of XML mapping files. In the NCAM project, the Netbeans-integrated development environment with the Hibernate Java persistence API plug-in includes the necessary Annotations package.

For Hibernate Annotations, one needs to annotate the getter methods of all attributes that need to be persisted. By default, all primitive data types are persisted and references need to be annotated with their cardinality with their associating classes. If one wishes to override the default behavior for database field names, primitive attribute names may be overridden for their corresponding database field column names. References that are not persisted are annotated as “@Transient” on the getter method. The reader may need to read the Hibernate Annotations documentation to comprehend all the intricacies of annotations.

Looking at the pertinent portions of the AntennaModel class (Markowski et al., 2012) with the code in italics and two forward slashes precede the code discussion comments, the class is annotated as follows. The code shows some basic annotations of a class that needs to be persisted.

The class is annotated as an Entity to identify to Hibernate that these objects may need to be persisted to a relational database.

@Entity
public class AntennaModel implements NCAMModel, Serializable {
boolean tracking;
..

//Each Entity class needs a primaryKey attribute declared Long, so that a relational database may identify each record as unique via the primary key. Each row in a database represents one entity object or one record, in database terms.

Long primaryKey;

//Reference “rm” to be annotated to identify to Hibernate the multiplicity between AntennaModel and RadioModel as many: one

RadioModel rm;

//Annotated to identify to Hibernate that the attribute “primaryKey” is the designated database primary key for each persisted object or record in the database table. @GeneratedValue identifies to Hibernate to track the database primaryKeys by properly incrementing, deleting it, and keeping the primary keys unique as required in relational databases.

```
@Id
   @GeneratedValue
   public Long getPrimaryKey() {
      return primaryKey;
   }
..
```

//A many to one annotation is inserted to identify to Hibernate that this reference has a many to one multiplicity with the Radio Model, since there can be more than one antenna associated with each radio model object. The join column annotation identifies the foreign key in the AntennaModel database table and will label the foreign key column as “radioModel_id.”

```
@ManyToOne
   @JoinColumn(name = "RadiansioModel_id")
   public RadioModel getRm() {
      return rm;
   }…
}
```

9.2 Example of a Save or Update Database Operation for the AntennaModel.java Entity

One of the basic database operations is to save or update to a database with a new or revised record. For the AntennaModel.java code (Markowski et al., 2012), the following method is coded showing comments on implementation of a Hibernate DAO/factory design pattern for doing a save or update. Hibernate keeps track of which objects need to be saved or updated.

```
// The method is public returns void, since nothing will be returned.
public void dbSaveOrUpdate() {

//A factory object is created by calling the static getFactory method on the abstract DAOFactory class.
      DAOFactory factory = DAOFactory.getFactory();

// a database transaction is started which creates a new database transaction session.
      factory.getAntennaModelDAO().beginTransaction();

//antennaModel DAO object is retrieved
      AntennaModelDAO antennaModelDAO = factory.getAntennaModelDAO();

//the antennaModel object is saved into the DB
```

```
        antennaModelDAO.saveOrUpdate(this);

//the transaction is committed to the database
        factory.getAntennaModelDAO().commitTransaction();
      // the database session is closed
 factory.getAntennaModelDAO().closeSession();
    }
```

This code snippet shows a fairly simple example of a saveOrUpdate( ) method for database interactions via DAOs. All references to Hibernate are hidden from the Java programmer, and only OO principles need to be understood. Similarly, other database CRUD operations are Java-coded for each Entity that needs to interact with a relational database.

10. NCAM Increased Database Read Speed Implementation for Antenna Patterns

Reading complex three-dimensional radio antenna patterns from a database is a time-intensive database operation. Since a typical U.S. Army wireless communication scenario may have hundreds of radios with identical antenna patterns, a read only once for unique antenna patterns from the database was incorporated into the database architecture of the NCAM software. The design concept is to read all unique antenna patterns used in a scenario once and store the antenna patterns in an array. As an antenna pattern is needed when NCAM builds its scenario in memory, the antenna pattern is pulled from the array instead of reading it from the database.

The design implementation is incorporated into the AntennaList.java class in the antenna package of the NCAMLib project and follows the design concept of reading the antenna patterns into an array and dealing with multithreading using “double-checked locking” to reduce the use of synchronization in the getInstance() method (Freeman et al., 2004, pp. 180–182).

With double-checked locking, we first check to see if an instance is created. Once inside the second “if” block, we check again, and if still null, we create an instance, i.e., *uniqueInstance = new AntennaPatternList();*

The following condensed source code shows this implementation:

```
public class AntennaPatternList {
    private volatile static AntennaPatternList uniqueInstance;

    List<AntennaPattern> listApl;

    private AntennaPatternList() {
```

```
    listApl=new LinkedList();
  }

  public static AntennaPatternList getInstance() {

    if (uniqueInstance == null) {
      synchronized (AntennaPatternList.class) {
        //double checked locking to see if Instance is created
        //if not then synchcronize - synchronize only first time thru
        if (uniqueInstance == null) {
          uniqueInstance = new AntennaPatternList();
        }
      }
    }

    return uniqueInstance;
  }
```

The volatile keyword in the example ensures that multiple threads handle the unique instance correctly when it is being initialized to the Singleton instance. The returned instance of uniqueInstance of a AntennaPatternList is known as a Singleton, an instantiation of a class to one object. The private constructor ensures that class may be instantiated only inside the class. The static method is invoked to create one instance, i.e., only one instance of AntennaPatternList.

For multi-threading purposes, synchronization is required, and double-checking is incorporated to reduce the overhead for the synchronization in the getInstance() method.

11. NCAM Database Speed Enhancements for Bulk Deletes of Database Records

Hibernate provides methods for executing SQL via HQL that do not affect in memory state of objects needing persistence. Hibernate as an object relation tool incorporates automatic and transparent object/relational mapping with management of object state. HQL provides for a way for manipulation of data directly in the database via the HQL/SQL constructs. When doing bulk deletes from the database, HQLs provide for a speed enhancement in deleting records that cannot be achieved via the DAO construct. NCAM incorporates this construct in the DeploymentResult.java, PropgationResult.java, and the remaining modules.

12. Conclusion

This report has identified and discussed the database architecture design concepts that are implemented into the NCAM software code. One of the time-saving features of Hibernate is the ability to generate database schemas from the domain model entities with a simple program incorporated into the NCAM software, i.e., NCAMLib-persist-CreateDBSSchema.java class. Running this file recreates the database schema in the database for any changes that have been made to the Entity persistence domain model without any additional SQL type of coding. Also, Hibernate's ability to address database interactions with an OO approach simplifies the programming effort and also decreased the source code size by about 50% from our initial assessments. This has saved a considerable amount of coding time since no tedious JDBC/SQL coding is required to generate a revised database schema but only the attributes the domain Entity model requires. Hibernate has the capability of implementing HQL or direct injection of SQL into the database, which provides increased speed for certain bulk database operations that are implemented in NCAM. A time-saving approach in developing database support for programs is to use Hibernate in a total OO sense, so that the domain entity model may be used to build the database schema. Once the database matures and there are a few changes in the domain model further efficiencies and speed enhancements may be implemented using HQL or SQL constructs. This approach was used in this project with significant time savings in the Java coding development process.

Appendices A and B describe address setup procedures for installing the MySQL database management system and identifying the database dictionary for the NCAM software.

13. References

1. Markowski, M.; Bevec, A.; Chike, N. Network Connectivity Analysis Model software comprised of the NCAM project Source Code Listing and NCAMLib project source code listing, U.S. Army Research Laboratory, Survivability/Lethality Analysis Directorate Communications, Electronic Warfare Branch, 18 August 2012.

2. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software;* Addison-Wesley Professional: Boston, MA,1994.

3. Hibernate Home Page. www.hibernate.org/about (accessed 15 February 2013).

4. MySQL Home Page. http://dev.mysql.com/downloads/mysql/ (accessed 2 March 2010).

5. McKenzie, C. *Hibernate Made Easy: Simplified Data Persistence with Hibernate and JPA Annotations;* PulpJava: Palo Alto, CA, 2008.

6. Freeman, E.; Freeman, E.; Bates, B.; Sierra, K.; Robson, E. *Head First Design Patterns;* O'Reilly Media Inc.: Sebastopol, CA, 2004.

INTENTIONALLY LEFT BLANK.

Appendix A. Scenario Database Schema and Data Dictionary

Hibernate maps the Network Connectivity Analysis Model (NCAM) Java Entity class model in the NCAMLib project to a “scenario” database schema. This is accomplished in the driver class CreateDBSchema.java class found in the persist package of the source code. Hibernate uses the HibernateUtil.recreateDatabase() method to recreate the database schema if the Entity model changes for NCAM. This is accomplished by running the CreateDBSchema.java class. Similarly, this is also accomplished for the BlueSystems and RedSystems databases. The following shows the Entity model.

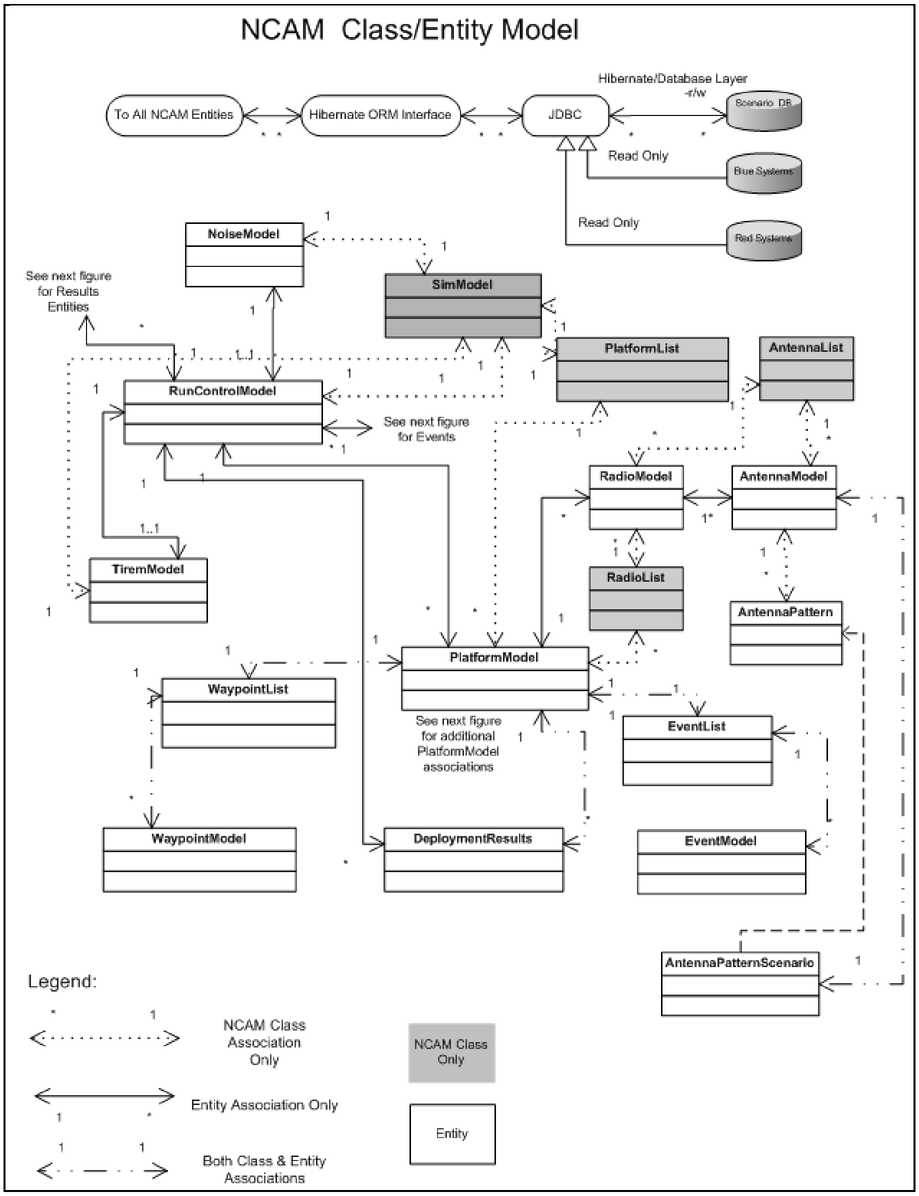
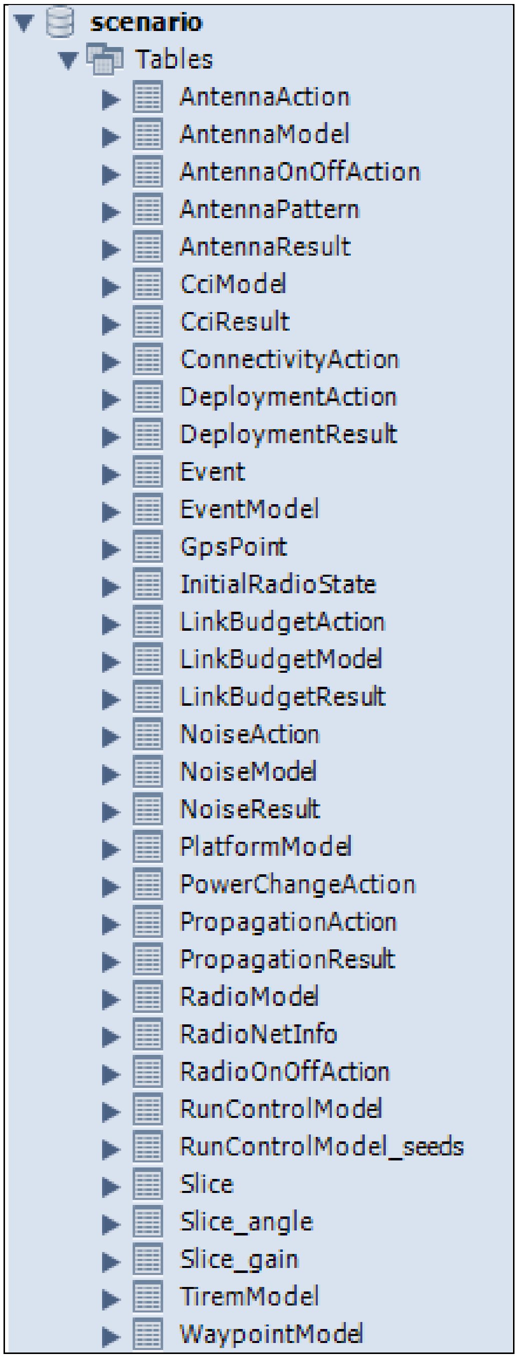


Figure A-1. NCAM Class/Entity model.

The Entity class model is converted to the following “scenario” database with its table layout. The data dictionary for the database schema follows:



The Scenario Database and Table Associations to the NCAM Modules.

Table Name	NCAM Module Association	Properties Captured in Table/Comments
AntennaAction	Antenna	Simulates the gain at each antenna in each link. Schedule time of antenna action and EventID.
AntennaModel	Antenna	Properties of antenna, i.e., model number, height, and antenna pattern associated.
AntennaOnOffAction	Antenna	Simulates antenna on or off action. Antenna name, radio name, schedule time, antenna on/off, and sector group name.
AntennaPattern	Antenna	Antenna pattern properties used by blue or red radio antennas. The patterns are read from the read-only blueSystems or redSystems databases and then stored also in the Scenario database. This is for users that wish to take all their scenario information, including antenna patterns, with them after completing NCAM runs. This Entity is not presently persisted into the Scenario database but into the blue or red databases. It is available here for possible future use.
AntennaResult	Antenna	Calculated antenna results, e.g., gains between antenna links i.e., gainRx, gainTx, bearing, azimuth, platform color, and event time.
CCIModel	Connectivity	Not persisted (may be used in the future).
CCIResult	Connectivity	Results for the connectivity confidence interval (CCI) for receiving radio, S/N for receiver, signal standard deviation (STD), signal and jammer STD, radio reciever threshold level, z value, etc.
ConnectivityAction	Connectivity	Scheduled time for Connectivity Action event.

The Scenario Database and Table Associations to the NCAM Modules (continued).

Table Name	NCAM Module Association	Properties Captured in Table/Comments
DeploymentResult	Deployment	Deployment module results, e.g., latitude, longitude, elevation, position.
Event	All modules, event type-dependent	Event time and primary key of action types: antenna action, noise action, propagation action, position interpolation action, and transceiver action .
EventModel	All modules	Event time and event model name.
GpsPoint	Deployment	Location in latitude/longitude, time, and speed in kilometers per hour (kph) of platform.
InitialRadioState	Initial radio state	Is the radio initially turned on?
LinkBudgetAction	Link budget	Scheduled time of LinkBudgetAction persisted for paused simulation.
LinkBudgetModel	Link budget	Persisted for possible future use.
LinkBudgetResult	Link budget	Values from LinkBudgetResult.
NoiseAction	Noise	Scheduled time of NoiseAction persisted for paused simulation.
NoiseModel	Noise	External natural noise and radio-generated noise values.
NoiseResult	Noise	Noise impinging on a radio node from natural to man-made noise.
PlatformModel	Deployment	Platform property types, i.e., urn, role, blue, or red.
PowerChangeAction	Deployment	Power level changes to the radio.
PropagationAction	Propagation	Scheduled time of PropagationtAction persisted for paused simulation.
PropagationResult	Propagation	Free space and path loss between platform/radio/antenna links at time; mode, i.e., diffraction or line of sight, antenna type.
RadioModel	Deployment, antenna	Radio properties.
RadioNetInfo	Deployment	Radio network properties.
RadioOnOffAction	Deployment	Scheduled time for radio on and platform association.
RunControlModel	Run control	Initial state of scenario, e.g., number of red and blue platforms; scenario name; start, stop, and step of simulation; transceiver or jammer on off; initial seed.
RunControlModel__seeds	Run control	Random seed control for stochastic events.
Slice	Antenna	Antenna pattern slice; each antenna pattern is composed of a number of azimuthal slices.
Slice_angle	Antenna	Elevation angle inside of slice.
Slice_gain	Antenna	Gain due to antenna pattern at set azimuthal slice and its corresponding elevation angle.

The Scenario Database and Table Associations to the NCAM Modules (continued).

Table Name	NCAM Module Association	Properties Captured in Table/Comments
TiremModel	Propagation	Environmental and electromagnetic propagation characters, e.g., conductivity, permittivity, frequency.
WaypointList	Deployment	Captures list of waypoint models associated with each platform model.
WaypointModel	Deployment	Waypoint properties of platform model as it traverses its path, i.e., latitude, longitude, time of arrival and departure, speed.

The Data Dictionary for Each Table Listed in Alphabetical Order.

AntennaAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table	NA	Long
scheduledTime	Scheduled time	Scheduled simulated clock time when an antenna action event occurs for starting a paused simulator	Seconds	double

AntennaModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
active	Active	Active antenna on system.	NA	boolean
azimuth	Azimuth angle	Azimuthal angle in degrees relative to vehicle.	Digital degrees	double
coordTrackingAntenna	Coordinate tracking antenna boolean	Does the antenna track another platform's coordinates for fixed coordinates?	NA	boolean
elevation	Elevation angle	Elevation angle relative to vehicle's forward direction.	Digital degrees	double
fixedAntenna	Fixed antenna boolean	Is this a fixed antenna on a platform?	NA	boolean
forceColor	Force color	Force color either "b" blue for friendly or "r" red for enemy.	NA	char
height	Height	Height of antenna above ground.	Meters	double
id	Identification (ID) number	ID number in graphic user interface (GUI).	NA	int
modelNumber	Model number	Model number of antenna for its respective antenna pattern; used to pull out the proper antenna pattern for this antenna model.	NA	String
name	Name	Antenna name other than model number.	NA	String
polarization	Polarization	Polarization of antenna pattern either vertical or horizontal.	NA	char
rxCableLoss	Receiver cable loss	Loss in decibels for the receiver cable.	Decibels	double
rxConnectorLoss	Receiver connector loss	Connector loss in decibels.	Decibels	double

AntennaModel (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
rxOtherLoss	Reciever other loss	Other receiver line losses not accounted from connector or line.	Decibels	double
rxRadomeLoss	Receiver radome loss	Loss due to radomesurrounding antenna.	Decibels	double
sectorGroup	Sector group	Sector group that the antenna belongs to.	NA	String
targetAntennaID	Target antenna ID number	Target ID of antenna being tracked by this antenna.	NA	int
targetBlue	Target blue	Is the target being tracked a blue platform?	NA	boolean
targetHeight	Target height	Target height above mean sea level in meters.	Meters	double
targetLatitude	Target latitude	Latitude of target.	Degrees	double
targetLongitude	Target longitude	Longitude of target.	Degrees	double
targetPlatformID	Target platform ID number	Target ID number of platform being tracked.	NA	int
targetRadioID	Target radio ID number	Target ID number of radio being tracked	NA	int
targetURN	Target URN	Target uniform resource number of platform being tracked.	NA	String
txCableLoss	Transmitter cable loss	Loss of radio transmitter cable.	Decibels	double
txConnectorLoss	Transmitter connector loss	Loss of cable connector.	Decibels	double

AntennaModel (continued)

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
txOtherLoss	Transmitter other loss	Other losses associated with radio platform while transmitting.	Decibels	double
txRadomeLoss	Transmitter radome loss	Loss due to radome surrounding antenna.	Decibels	double
usedForRx	Used for receiving	Is this antenna used for receiving signals, true or false?	NA	boolean
usedForTX	Used for transmitting	Is this antenna used for transmitting signals, true or false?	NA	boolean
vehTrackingAntenna	Vehicle tracking antenna	Is this a vehicle tracking antenna, true or false?	NA	boolean
radioModel_id	Radio model foreign key ID number	Foreign key for pairing this antenna model with the radio's primary key.	NA	Long

AntennaOnOffAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
antennaName	Antenna name	The antenna name associated with the antenna.	NA	String
antennaOn	Antenna on	Is the antenna on at a particual time?	NA	boolean
blue	Blue	Is the antenna associated with a blue platform?	NA	boolean
netName	Network name	The network or subnet that the antenna is associated.	NA	String
platformId	Platform identification (ID) number	Each platform has a unique ID number. This is not a the primary key, but a number used in the source code.	NA	int
radioName	Radio name	Name associated with radio.	NA	String
scheduleTime	Schedule time	Time schedule when the antenna is turned on or off from Epoch time 1 January 1970, converted to milliseconds.	Milliseconds	long
sectGroupName	Section group name where the antenna belongs	Name of section that the antenna belongs to.	NA	String
event_id	Event ID number	Foreign key link to event.	NA	Long

AntennaPattern

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
frequencyIndex	Frequency index	The frequency index for multiple frequencies of a given model, presently not used, for future use if antenna patterns are given for various frequencies of a physical antenna.	None	int
modelNumber	Model number	The model number/name for a unique antenna.	None	String
nsma	National Spectrum Manager Association (NSMA)	Is this an NSMA antenna pattern?	NA	boolean
patNum	Pattern number	An NSMA-unique pattern number; not used; maintained for possible future use.	NA	String
polarization	Polarization	Polarization of antenna under test for the first character value, i.e., H or V; second is the source polarization; e.g., V/H vertical under test antenna/H is horizontal source antenna. Due to TIREM constraints, only V/V or H/H are currently used.	NA	String
rcm_id	Run control model ID number	Foreign key, linking this record with its run control model.	NA	Long

Note: TIREM = Terrain Integrated Rough Earth Model and a registered trademark of Alion Science and Technology, McLean, VA; H = horizontal linear; V = vertical linear.

AntennaResult

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
antRx	Antenna receiver	Antenna ID number for the receiver antenna.	NA	int
antTx	Antenna transmitter	Antenna ID number for the transmitter antenna.	NA	int
bearingAzimuth	Bearing azimuth	Azimuth bearing from transmitter antenna, antTx; bore sight to receiver antenna, antRx.	Radians	double
bearingElevation	Bearing elevation	Elevation bearing from transmitter antenna, antTx; bore sight to receiver antenna, antRx.	Radians	double
blueRx	Blue receiver	Is receiver antenna blue or friendly?	NA	boolean
blueTx	Blue transmitter	Is transmitter antenna blue or friendly?	NA	boolean
eventTime	Event time	Event time in milliseconds since Epoch time.	Milliseconds	long
gainRx	Gain of receiver	Gain at the receiver, in decibels isotropic.	Decibels isotropic	double
gainTx	Gain of transmitter	Gain at the transmitter, in decibels isotropic.	Decibels isotropic	double
idTx	Identification transmitter	Identification number of transmit platform, idTx; for link between idTx and idRx.	NA	int
idRx	Identification receiver	Identification number of receive platform, idRx; for link between idTx and idRx.	NA	int

AntennaResult (continued)

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
radioTx	Radio transmitter	Radio identification number for transmit radio and radioTx; for link between radios, radioTx, and radioRx.	NA	int
radioRx	Radio receiver	Radio identification number for receive radio and radioRx; for link between radios, raduiTx0, and radioRx.	NA	Int
t	Time	Antenna Result time for simulation snapshot time from Epoch time 1 January 1970.	Milliseconds	long
rcm_id	Run control model ID number	Foreign key, linking this record with its run control model.	NA	Long

CCIModel (persisted for future use)

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
runControlModel_id	Run control model ID	Foreign key.	NA	int

CCIResult

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
antRx	Antenna receiver identification number	Identification number.	NA	int
antTx	Antenna transmitter identification number	Identification number.	NA	int
blueRx	Blue receiver	Is it a blue force receiver?	NA	boolean
blueTx	Blue transmitter	Is it a blue force transmitter?	NA	boolean
CCI	Connectivity confidence interval	Captures the CCI value for tx to rx link.	Decibels	double
idRx	Identification number receiver	Identification number.	NA	intt
idTx	Identification number transmitter	Identification number.	NA	int
jamerStdDev	Jammer standard deviation	Jammer standard deviation for the propagation mode to receiver per TIREM values.	Decibels	double
radioRx	Radio reciever identification number	Identification number.	NA	int

CCIResult (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
radioTx	Radio transmitter identification number	Identification number.	NA	int
sAndJStdDev	Signal and jammer standard deviation	Combined standard deviation for propgation mode calculated as RMS value for the jammer and signal.	Decibels	double
sigStdDev	Signal standard deviation	Signal mode of propagation standard deviation from TIREM.	Decibels	double
signalToNoise	Signal to noise	S/N ratio at the receiver.	Decibels	double
threshold	Threshold	S/N decibels at 50% packet completion threshold of radio.	Decibels	double
t	Time	CCI result time for simulation snapshot time from Epoch time 1 January 1970.	Milliseconds	long
z	Z value	How many standard deviations an observation or datum is above or below the mean?	NA	double

Note: S/N = signal-to-noise ratio; RMS = root-mean-square.

CCIResult (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
rcm_id	Run control model ID#	Run control model foreign key for database purposes.	NA	Long

ConnectivityAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
scheduledTime	Scheduled time	Current scheduled time for simulation snapshot time from Epoch time 1 January 1970.	Milliseconds	long
event_id	Event ID	Foreign key linking to parent event.	NA	Long

DeploymentAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
scheduledTime	Scheduled time	Current scheduled time for simulation snapshot time from Epoch time 1 January 1970.	Milliseconds	long
event_id	Event ID	Foreign key linking to parent event.	NA	Long

DeploymentResult

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
blue	Blue	Is platform blue?	NA	boolean
elevation	Elevation	Elevation is height above sea level for platform.	Meters	double
fwdX	Forward x	Platform’s forward pointing unit vector; unit vectors are the local coordnate system axes of the platform, value along x-axis.	NA	double
fwdY	Forward y	Platform’s forward-pointing unit vector, y-axis.	NA	double
fwdZ	Forward z	Platform’s forward-pointing unit vector, z-axis.	NA	double
id	Identification	ID number of platform assigned in GUI.	NA	int
latitude	Latitude	Latitude of platform.	Digital degrees	double
longitude	Longitude	Longitude of platform.	Digital degrees	double
posX	Position x	Platform’s Cartesian coordinate position; Cartesian coordinates in NCAM have origin at earth center, z-axis through north pole, x-axis through Prime Meridian, y-axis per right hand rule.	NA	double
posY	Position y	Platform’s Cartesian position.	NA	double
posZ	Position z	Platform’s Cartesian position.	NA	double
rightX	Right x	Platform’s right-pointing unit vector.	NA	double
rightY	Right y	Platform’s right-pointing unit vector.	NA	double

DeploymentResult (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
rightZ	Right z	Platform's right-pointing unit vector.	NA	double
t	Time	Time for this deployment result for simulation from Epoch time 1 January 1970.	Milliseconds	long
upX	Up x	Platform's upward-pointing unit vector, x value.	NA	double
upY	Up y	Platform's upward-pointing unit vector, y value.	NA	double
upZ	Up z	Platform's upward-pointing unit vector, z value.	NA	double
platformModel_id	Platform model ID	Foreign key linking to the platform model.	NA	long
rcm_id	Run control model ID	Foreign key linking to the run control model.	NA	long

Event

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
eventTime	Event time	Time that event should be executed from Epoch time, 1 January 1970.	Milliseconds	long
aa_primaryKey	Antenna action primary key	Foreign key for linking this row to AntennaAction table.	NA	Long
aof_primaryKey	Antenna on/off action primary key	Foreign key for linking this row to AntennaOnOffAction table.	NA	Long
ca_primaryKey	Connectivity action primary key	Foreign key for linking this row to ConnectivityAction table.	NA	Long
la_primaryKey	Link budget action primarykey	Foreign key for linking this row to LinkBudgetAction table.	NA	Long
na_primaryKey	Noise action primary key	Foreign key for linking this row to the NoiseAction table.	NA	Long
pa_primaryKey	Propagation action primary key	Foreign key for linking this row to the PropagationAction table.	NA	Long
pca_primaryKey	Power change action primary key	Foreign key for linking this row to the PowerChangeAction table.	NA	Long
pia_primarykey	Position interpolation primary key	Foreign key for linking this row to the PositionInterpolationAction table.	NA	Long

Event (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
rcm_id	Run control model ID	Foreign key for linking to RunControlModel table.	NA	Long
ta_primaryKey	Transceiver action primary key	Foreign key for linking this row to the TransceiverAction table.	NA	Long

EventModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
description	Description	Description of Event being saved and state.	NA	String
eventSaved	Event saved	Description of even type.	NA	String
eventTime	Event time	Time for scheduled event.	Seconds from Epoch time	double
platform_id	Platform ID number	Foreign key connecting event to platform model.	NA	long

GpsPoint

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
latitude	Latitude	Latitude of GPS unit.	Decimal degrees	double
longitude	Longitude	Longitude of GPS unit.	Decimal degrees	double
speed	Speed	Speed of GPS unit mounted on platform.	Kilometers/hour	double
t	Time	Unix epoch time since 00:00:00 coordinated universal time on 1 January 1970; captures time of data capture.	Milliseconds	long
platform_id	Platform ID number	Foreign key connecting GpsPoint to platform model.	NA	Long

InitialRadioState

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
initiallyOn	Initially on	Is the radio initially on at the start of the scenario?	NA	boolean
name	Name	Name of radio.	NA	String
rcm_id	Run control model ID	Foreign key for linking to RunControlModel table.	NA	Long

LinkBudgetAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
scheduledTime	Scheduled time	Current scheduled time for simulation snapshot time from Epoch time 1 January 1970.	Milliseconds	long
event_id	Event ID	Foreign key linking to parent event.	NA	long

LinkBudgetModel (unused)

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long

LinkBudgetResult

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
blueRx	Blue receiver	Is it blue receiver?	NA	boolean
blueTx	Blue transmitter	Is it blue transmitter?	NA	boolean
bwCorrectionFactor_dB	Band width correction factor	Ratio of radio noise equivalent bandwidth/jammer bandwidth.	Decibels	double
Eirp_dBm	Effective radiated power	Effective radiated power.	Decibel milliwatts	double
idAntennaRx	ID # of antenna on receiver	ID number of receiver antenna.	NA	int
idAtnennaTx	ID # of antenna on transmitter	ID number of transmitter antenna.	NA	int
idPlatformRx	ID # of platform recevier	ID number of receiver platform.	NA	int
idPlatformTx	ID # of platform transmitter	ID number of transmitter platform.	NA	int
idRadioRx	ID # of radio reciever	ID number of receiver radio.	NA	int
idRadioTx	ID # of radio transmitter	ID number of transmitter radio.	NA	int
jammerPower_W	Jammer power in watts	Power wattage of jammer in watts.	Watts	double
Signal_dBm	Signal dBm	Signal strength in decibel milliwatts.	Decibel milliwatts	double
snr	Signal to noise ratio	S/N ratio in decibels.	Decibels	double
t	Time	Time deployment result from Epoch time.	Seconds	long
totalNoise	Total noise	Total noise for all environments.	Decibel milliwatts	double
txPwr_dBm	Transmitter power	Transmitter power in decibel milliwatts.	Decibel milliwatts	double
rcm_id	Run control model ID	Run control model foreign key.	NA	Long

NoiseAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
scheduledTime	Scheduled time	Scheduled simulated clock time when an noise action event occurs for starting a paused simulator.	Seconds	double
event_id	Event ID	Foreign key that ties this action to the event.	NA	double

NoiseModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
cosmicRadtion	Cosmic radiation	Cosmic radiation contributing to noise; default is 2.725 from big bang remnant.	Degrees Kelvin	double
earthHeating	Earth heating	Earth heating background noise of 290 °K default.	Degrees Kelvin	double
moonLatitude	Moon latitude	Moon latitude above antenna contributing to background noise.	Decimal degrees	double
moonLongitude	Moon longitude	Moon longitude above antenna contributing to background noise.	—	—
noiseBandwidth	Noise band width	Radio noise bandwidth.	Hz	double
noiseFigure	Noise figure	Noise figure for radio.	Decibels	int
receiverTemp	Receiver temperature	Receiver temperature from noise module input (input as Fahrenheit stored in degrees Kelvin).	Degrees Kelvin	double
regionalNoise	Regional noise	Category of noise, i.e., rural, suburban, urban.	NA	String
rural	Rural	Is the platform in a rural environment?	NA	boolean
suburban	Suburban	Is platform in a suburban environment?	NA	boolean
sunLatitude	Sun latitude	Latitude of Sun above antenna.	Decimal degrees	double
sunLongitude	Sun longitude	Longitude of Sun above antenna.	Decimal degrees	double

NoiseModel (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
urban	Urban	Is platform in a urban environment?	NA	boolean
runControlModel_id	Run control model ID	Foreign key that links this record to the run control model.	NA	Long

NoiseResult

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
blue	Blue	Is it a blue receiver?	NA	boolean
cosmicNoise_K	Cosmic noise degrees Kelvin	Cosmic background noise in degrees Kelvin.	Degrees Kelvin	double
earthNoise_K	Earth noise in degrees Kelvin	Earth background noise.	Degrees Kelvin	double
externalNoise_K	External noise in decibel milliseconds	External noise in decibel milliseconds.	Decibel milliseconds	double
galacticNoise_dBm	Galactic noise in decibel milliseconds	Galactic background noise in decibel milliseconds	Decibel milliseconds	double
Id	ID #	ID # of platform.	NA	int
idRadio	ID # of radio	ID # of radio.	NA	int
internalNoise_dBm	Internal noise	Internal noise of radio.	Decibel milliseconds	double
noiseFigure_dBm	Noise figure in decibel milliseconds	Noise figure of radio.	Decibel milliseconds	double
receiverNoiseTemp_K	Receiver noise temperature in degrees Kelvin	Receiver noise temperature.	Degrees Kelvin	double
ruralNoise_dBm	Rural noise in decibel milliseconds	Rural noise.	Decibel milliseconds	double
suburbanNoise_dBm	Surburban noise in decibel milliseconds	Suburban noise.	Decibel milliseconds	double
t	Event time	Event time when noise occurs stored from Epoch time	Seconds	double
totalNoise_dBm	Total noise in decibel milliseconds	Total noise value in decibel milliseconds.	Decibels	double
urbanNoise_dBm	Urban noise in decibel milliseconds	Urban noise.	Decibel milliseconds	double
rcm_id	Run control model ID	Foreign key linking this record with the RunControlModel table.	NA	Long

PlatformModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
affiliation	Affiliation	Platform affiliation to upper level groups.	NA	String
Blue	Blue	Is the platform friendly, i.e., blue?	NA	boolean
bumperNum	Bumper number	Platform ID number.	NA	string
id	ID	ID number for GUI purposes.	NA	int
ipAddr	Internet provider (IP) address	IP address of platform.	NA	String
nodeNumber	Node number	Node number to keep track for some user requirements.	NA	String
numWaypoints	Number of waypoints	Number of waypoints from beginning to the end of the simulation.	NA	int
platformClass	Platform class	Ground, air, sea, space.	NA	String
rolePlatform	Platform role	What role does the platform play, e.g., commander, leader, relay.	NA	String
type	Type	Type of platform, e.g., jammer, platform description.	NA	String
urn	Uniform resource number	Unique identification used in the U.S. Army, for each platform, similar to serial number.	NA	String
rcm_id	Run control model identification number	Foreign key, for pairing the run control model primary key with this record, i.e., this foreign key = primary key of run control model record.	NA	Long

PowerChangeAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
blue	Blue	Is power change action for blue radio, i.e., friendly?	NA	boolean
netName	Network name	Name of network the radio belongs to.	NA	String
platformID	Platform ID number	Platform ID number used by NCAM to track platform.	NA	int
powerLevel	Power level	Power level of radio.	Watts	double
radioName	Radio name	Name of the radio.	NA	String
scheduledTime	Scheduled time	Scheduled simulated clock time when a power change action event occurs for starting a paused simulator.	Seconds	double
event_id	Event ID	Foreign key that ties this action to the event.	NA	double

PropagationAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
scheduledTime	Scheduled time	Scheduled simulated clock time when a propagation action event occurs for starting a paused simulator.	Seconds	double
event_id	Event ID	Foreign key that ties this action to the event.	NA	double

PropagationResult

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
antRx	Antenna receiver	Receiver antenna ID number from user input.	NA	int
antTx	Antenna transmitter	Transmitter antenna ID number from user input.	NA	int
azimuth	Azimuth	Azimuth of platform Tx or idTx to platformRx or idRx; the angle with a line from Tx pointing at true north and to platformRx or idRx, measuring counter clockwise as positive.	Decimal degrees	double
blueRx	Blue receiver	Is the receiver a blue receiver?	NA	boolean
blueTx	Blue transmitter	Is the transmitter radio blue?	NA	boolean
freeSpaePathLoss	Free space path loss	Free space loss in decibels.	Decibels	double
idRx	ID receiver	Receiver platform ID number.	NA	int
idTx	ID transmitter	Transmitter platform ID number.	NA	int
mode	Mode	Mode of propagation loss, i.e., line of sight or diffraction.	NA	String

PropagationResult (continued).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
pathLoss	Path loss	Path loss in decibels.	Decibels	double
radioRx	Radio receiver	Receiver radio ID number.	NA	int
radioTx	Radio Transmitter	Transmitter radio ID number.	NA	int
takeoffRx2D	Take off receiver 2-D	Take off angle of receiver in two dimensions (2-D).	Radians	double
takeoffTx2D	Take off transmitter 2-D	Take off angle of transmitter in 2-D.	Radians	double
takeoffRxX	Take off angle of receiver X direction	Take off angle of receiver in unit vector form -X direction.	Meters	double
takeoffRxY	Take off angle of receiver Y direction	Take off angle of receiver in unit vector form -Y direction.	Meters	double
takeoffRxZ	Take off angle of receiver Z direction	Take off angle of receiver in unit vector form -Z direction.	Meters	double
takeoffTx2D	Take off transmitter 2-D	Take off angle of transmitter in 2-D.	Radians	double
takeoffTxX	Take off angle of transmitter X direction	Take off angle of transmitter in unit vector form -X direction.	Meters	double
takeoffTxY	Take off angle of transmitter Y direction	Take off angle of transmitter in unit vector form -Y direction.	Meters	double
takeoffTxZ	Take off angle of transmitter Z direction	Take off angle of transmitter in unit vector form -Z direction.	Meters	double
t	Time	Time of snapshot for captured deployment result.	Seconds	double
rcm_id	Run control model ID	Foreign key linking the record in propagation result with the run control model primary key.	NA	Long

RadioModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
bandwidth	Bandwidth	Bandwidth of radio waveform.	Megahertz	double
bw60dB	Bandwidth 60 dB	Radio bandwidth at (-60dB).	Megahertz	double
bw60dB	Bandwidth 6 dB	Radio bandwidth at (-6dB).	Megahertz	double
frequency	Frequency	Frequency of radio.	Megahertz	double
id	ID number	ID number of radio set in GUI.	NA	—
insertionLoss	Insertion loss	Insertion loss in selectivity.	Decibels	double
modulationName	Modulation name	Modulation name of radio waveform.	NA	String
name	Name	Name of radio, identified as type in GUI.	NA	String
net	Network	Network radio belongs to.	NA	String
noiseEquivalentBandwidth	Noise equivalent bandwidth	Noise equivalent bandwidth of radio.	Megahertz	double
noiseFigure	Noise figure	Noise figure for radio.	Decibels	double
power	Power	Radio power.	Watts	double
radioOn	Radio on	Is the radio on initially?	NA	boolean
sensitivity	Sensitivity	Sensitivity of radio.	Decibel milliseconds	double
threshold	Threshold	Signal to noise lab determine threshold of 50% packet completion rate.	Decibels	double
platformModel_id	Platform model ID number	Foreign key linking radio model record with platform model primary key.	NA	Long

RadioNetInfo

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
name	Name	Name of radio.	NA	String
net	Network	Name of network.	NA	String
rigOn	Rig on	Is the network active?	NA	Boolean

RadioOnOffAction

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
blue	Blue	Is action for blue radio?	NA	boolean
netName	Network name	The name of the network radio is connected to.	NA	String
platformId	Platform identification	Platform identification number.	NA	int
radioName	Radio name	Name of radio.	NA	String
radioOn	Radio on	Is the radio on initially?	NA	boolean
scheduledTime	Scheduled time	Scheduled simulated clock time when a radio on/off action event occurs for starting a paused simulator.	Seconds	double
event_id	Event ID	Foreign key that ties this action to the event.	NA	double

RunControlModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
distanceAir	Distance air	Maximum air distance for radio communication when recalculation occurs.	Meters	double
distanceGround	Distance ground	Maximum ground distance for radio communication when recalculation occurs.	Meters	double
firstSampleTime	First sample time	Time in seconds when scenario is sampled.	Seconds	double
lastSampleTime	Last sample time	Time in seconds when the last the scenario is sampled.	Seconds	double
numReplication	Number of replications	How many times does one wish to rerun the scenario with different random seeds? Default is one.	NA	Int
runDate	Run date	Date run was executed.	NA	String
sampleTimeStep	Sample time step	The snapshot time interval to sample the NCAM run.	Milliseconds	double
scenarioName	Scenario name	The name of the run scenario.	NA	String
subVersionControlNo	Subversion control number	Subversion control number used to track NCAM code freeze.	NA	String
useWgs84	Use WGS84	Use the WGS84 coordinate system.	NA	boolean
username	User name	User name of NCAM scenario run.	NA	String
noiseModel_id	Noise model ID	Foreign key, linking noise model to run control model.	NA	Long
tiremModel_id	TIREM model	Foregin key linking TIREM model with run control model.	NA	Long

RunControlModel_seeds

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
RunControlModel_primayKey	Run control model primary key	Captures primary key for each record/row in table; it is mapped by Hibernate as an array/table generated in the RunControlModel.java class.	NA	Long
SeedNumber	Seed number	Random seed number generated by pseudo random key generator; needs to be different or the same depending on results wishing to obtain if running replications of the same scenario.	—	—
seed_index	Seed index	Seed index in the index of the array holding the seed.	NA	int

Slice (not used in Scenario database – retained for possible future use).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
azimuth	Azimuth	Azimuth angle of slice for a 3-D antenna pattern. The azimuth angle progresses counterclockwise starting at north=0 rad, i.e., math notation.	Radians	double
maxAngle	Maximum angle	Maximum angle that slices are available for the 3-D antenna pattern; usually 6.28 rad, i.e., 2 Pi.	Radians	double
minAngle	Minimum angle	Minimum angle that slices are available for the 3-D antenna pattern; usually 0 rad.	Radians	double
AntennaPattern_id	Antenna Pattern ID number	Foreign key for linking this table to the antenna pattern table.	NA	Long

Slice_angle (not used in Scenario database – retained for possible future use).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
Slice_primaryKey	Foreign key linking this entry to the Slice	Captures foreign key for each row linking this entry to the slice table; since Hibernate generates these keys from an array persistence model, no primary keys are generated; only foreign keys.	NA	Long
angle	Elevation angle within the slice	Elevation angle starting at 0 rad or horizontal to the tangent to the earth's surface, and progressing upward.	Radians	double
angle_index	Angle index	Hibernate generates an index for each elevation entry row. This corresponds to the same index as in the Slice_gain table. Thus, as the elevation progresses through each index, so do the corresponding values in the Slice_gain table. The Slice_primaryKey and the index link the values in this table to a unique slice.	NA	long

Slice_gain (not used in Scenario database – retained for possible future use).

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
Slice_primaryKey	Foreign key linking this entry to the slice	Captures foreign key for each for linking this entry to the slice table; since Hibernate generates these keys from array persistence models, no primary keys are generated; only foreign keys.	NA	Long
gain	Gain	Gain from antenna pattern at the slice, and elevation angle identified by the gain_index and Slice_prmaryKey.	Decibels	double
gain_index	Gain index	Hibernate generates an index for each gain row entry. This corresponds to the same index as in the Slice_angle table. The Slice_primaryKey and the gain index link the values in this table to a unique slice.	NA	long

TiremModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
conductivity	Conductivity	Conductivity.	siemens (S)/m	Float
frequency	Frequency	Frequency of propagating wave.	Megahertz	double
humidity	Humidity	Humidity of atmosphere.	g/m**3	Float
interpolationType	Interpolation type	Interpolation type required in TIREM; default “nearest.”	NA	String
lossFraction	Loss fraction	Percentage of year path loss is not exceeded.	Percentage	float
permittivity	Permittivity	Relative permittivity from 1–100; default = 15.	NA	float
refractivity	Refractivity	200–400 N units; default = 295.	NA	float
rcm_id	Run control model id	Foreign key pairing this record with run control model primary key.	NA	Long

WaypointModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
accelerate	Accelerate	Get platform's acceleration from blue database; default = 2.0.	m/s**2	double
allWps	All waypoints	Total number of waypoints in scenario.	NA	int
arrivalSpeed	Arrival speed	Arrival speed of platform at destination waypoint.	Meters/second	double
arrivalTime	Arrival time	Arrival time of platform at destination waypoint.	Seconds	double
changed	Changed	Unused.	—	—
cruiseSpeed	Cruise speed	Platform cruising speed.	Meters/second	double
departTime	Depart time	Time platform departs destination waypoint.	Seconds	double
dist	Distance	Distance between two consecutive way points.	Meters	double
elevation	Elevation	Platform's elevation above sea level.	Meters	double
initDepartTime	Initial depart time	Depart time at initial way point.	Seconds	double
initSpeed	Initial speed	Arrival speed at initial way point	Seconds	double
initTime	Initial time	Arrival time at initial way point.	Seconds	double
isChangedLat	Is changed latitude	Has latitude changed for platform compared to previously?	NA	boolean
isChangedLong	Is changed longitude	Has longitude changed for platform compared to previously?	NA	boolean
k	k	Ratio of altitude/to distance traversed.	NA	double
latitude	Latitude	Current latitude of platform.	Decimal percentage	double
loiterSec	Loiter seconds	Time platform loiters at waypoint.	Seconds	double
loiterSpeed	Loiter speed	Loiter speed of aircraft above a certain loiter point.	Seconds	double
longitude	Longitude	Current longitude of platform.	Decimal °	double
maxCruise	Maximum cruise	Maximum cruising speed of platform to be read in from blue database; default = 20.	Meters/second	double

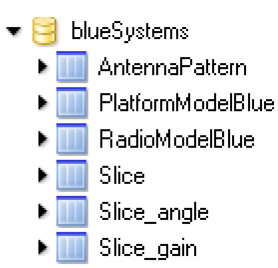
WaypointModel

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
maxVel	Maximum velocity	Maximum speed of platform to be read in from blue database; default = 20.	Meters/second	double
minVel	Minimum velocity	Minimum speed of platform to be read.	Meters/second	double
t1	Time 1	Time it takes platform to accelerate to cruising speed.	Seconds	double
t2	Time 2	Time it takes platform to cruise.	Seconds	Double
wayPt	Way point	Platform's current way point.	NA	int
platformModel_ID	Platform model ID number	Foreign key tying this record waypoint with platformModel.	NA	Long

The "blueSystems" Database Tables and the Association to NCAM Modules

Table Name	NCAM Module Association	Properties Captured in Table/Comments
AntennaPattern	Antenna	Antenna pattern properties used by blue radio antennas.
PlatformModelBlue	Deployment	Blue platforms that are presented in the platforms tab of the deployment module.
RadioModelBlue	Deployment	Radio properties of blue/friendly forces.
Slice	Antenna	Antenna pattern slice, each antenna pattern is composed of a number of azimuth slices.
Slice_angle	Antenna	Elevation angle inside of slice.
Slice_gain	Antenna	Gain due to antenna pattern at set azimuth slice and its corresponding elevation angle.

blueSystems Database Table Layout



The Data Dictionary for Each Table in the Blue Database in Alphabetical Order

AntennaPattern

Database FieldName/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
frequencyIndex	Frequency index	The frequency index for multiple frequencies of a given model, presently not used, for future use if antenna patterns are given for various frequencies of a physical antenna.	None	Int
modelNumber	Model number	The model number/name for a unique antenna.	None	String
polarization	Polarization	Polarization of antenna under test for the first character value, i.e., H or V; and the second is the source polarization; e.g., V/H vertical under test antenna/H is horizontal source antenna. Due to TIREM constraints, only V/V or H/H are currently used.	NA	String

PlatformModelBlue

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
affiliation	Affiliation	Platform affiliation to upper level groups.	NA	String
antenna	Antenna	General antenna category, e.g., whip, vertical.	NA	String
antennaHeight	Antenna height	Antenna height from base to tip.	Meters	double
forcePlatform	Force platform	Force of platform, i.e., red or blue.	NA	String
frequency	Frequency	Frequency of radio.	Megahertz	double
ipAddr	IP address	IP address of platform.	NA	String
platformClass	Platform class	Class of platform, i.e., ground, air, space, water. Value extracted from blueDB; only ground available currently.	NA	String
polarization	Polarization	Polarization of antenna, only V or H available.	NA	char
power	Power	Power of radio.	Watts	double
radio	Radio	Radio model name, e.g., SINCGARS.	NA	String
radioOn	Radio on	Is the radio on?	NA	boolean
rolePlatform	Platform role	What role does the platform play, e.g., commander, leader, relay?	NA	String
type	Type	Type of platform, e.g., jammer, platform description.	NA	String
urn	Uniform resource number	Unique identification used in the U.S. Army, for each platform, similar to serial number.	NA	String

Note: SINCGARS = Single Channel Ground and Airborne Radio System.

RadioModelBlue

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
bandwidth	Bandwidth	Bandwidth of radio waveform.	Megahertz	double
bw60db	Bandwidth 60 dB down	In selectivity, stopband at (-60dB).	Megahertz	double
bw6db	Bandwidth 6 Db down	In selectivity, passband at (-6dB).	Megahertz	double
frequency	Frequency	Frequency of radio.	Megahertz	double
insertionLoss	Insertion loss	Insertion loss in selectivity.	Decibels	double
noiseEquivalentBandwidth	Noise equivalent bandwidth	Noise equivalent bandwidth of radio.	Megahertz	double
noiseFigure	Noise figure	Noise figure for radio.	Decibels	double
power	Power	Radio power.	Watts	double
radio	Radio	Radio name as seen by user in deployment setup; radios tab "Type" combo selection. Values are shown from blue database, e.g., SINCGARS, EPLRS.	NA	String
sensitivity	Sensitivity	Sensitivity of radio.	Decibel milliseconds	double

Note: EPLRS = Enhanced Position Location Reporting System.

Slice

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
azimuth	Azimuth	Azimuth angle of slice for a 3-D antenna pattern. The azimuth angle progresses counter-clockwise starting at north = 0 rad, i.e., math notation.	Radians	double
maxAngle	Maximum angle	Maximum angle that slices are available for the 3-D antenna pattern; usually 6.28 rad, i.e., 2 Pi.	Radians	double
minAngle	Minimum angle	Minimum angle that slices are available for the 3-D antenna pattern; usually 0 rad.	Radians	double
AntennaPattern_id	Antenna pattern ID number	Foreign key for linking this table to the antenna pattern table.	NA	Long

Slice_angle

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
Slice_primaryKey	Foreign key linking this entry to the slice	Captures foreign key for each row linking this entry to the slice table; since Hibernate generates these keys from an array persistence model, no primary keys are generated; only foreign keys.	NA	Long
angle	Elevation angle within the slice	Elevation angle starting at 0 rad or horizontal to the tangent to the Earth's surface, and progressing upward.	Radians	double
angle_index	Angle index	Hibernate generates an index for each elevation entry row. This corresponds to the same index as in the Slice_gain table. Thus, as the elevation progresses through each index, so do the corresponding values in the Slice_gain table. The Slice_primaryKey and the index link the values in this table to a unique slice.	NA	long

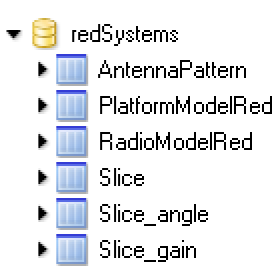
Slice_gain

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
Slice_primaryKey	Foreign key linking this entry to the slice	Captures foreign key for each for linking this entry to the slice table; since Hibernate generates these keys from array persistence models, no primary keys are generated; only foreign keys.	NA	Long
gain	Gain	Gain from antenna pattern at the slice, and elevation angle identified by the gain_index and Slice_prmaryKey.	Decibels	double
gain_index	Gain index	Hibernate generates an index for each gain row entry. This corresponds to the same index as in the Slice_angle table. The Slice_primaryKey and the gain index link the values in this table to a unique slice.	NA	long

The “redSystems” Database Tables and the Association to NCAM Modules

Table Name	NCAM Module Association	Properties Captured in Table/Comments
AntennaPattern	Antenna	Antenna pattern properties used by red radio antennas.
PlatformModelRed	Deployment	Red platforms that are presented in the platforms tab of the deployment module.
RadioModelRed	Deployment	Radio properties of red/enemy forces
Slice	Antenna	Antenna pattern slice, each antenna pattern is composed of a number of azimuth slices.
Slice_angle	Antenna	Elevation angle inside of slice.
Slice_gain	Antenna	Gain due to antenna pattern at set azimuth slice and its corresponding elevation angle.

redSystems Database Table Layout



The Data Dictionary for Each Table in the Red Database in Alphabetical Order.

AntennaPattern

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
frequencyIndex	Frequency index	The frequency index for multiple frequencies of a given model, presently not used, for future use if antenna patterns are given for various frequencies of a physical antenna.	None	int
modelNumber	Model number	The model number/name for a unique antenna.	None	String
polarization	Polarization	Polarization of antenna under test for the first character value, i.e., horizontal (H) or vertical (V); the second is the source polarization; e.g., V/H vertical under test antenna/H is horizontal source antenna. Due to TIREM constraints, only V/V or H/H are currently used.	NA	String

PlatformModelRed

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
affiliation	Affiliation	Platform affiliation to upper level groups.	NA	String
antenna	Antenna	General antenna category, e.g., whip, vertical.	NA	String
antennaHeight	Antenna height	Antenna height from base to tip.	Meters	double
forcePlatform	Force platform	Force of platform, i.e., red or blue .	NA	String
frequency	Frequency	Frequency of radio.	Megahertz	double
ipAddr	IP address	IP address of platform.	NA	String
platformClass	Platform class	Class of platform, i.e., ground, air, space, water; value extracted from redDB; only ground available currently.	NA	String
polarization	Polarization	Polarization of antenna, only V or H available.	NA	char
power	Power	Power of radio.	Watts	double
radio	Radio	Radio model name, e.g., SINCGARS.	NA	String
radioOn	Radio on	Is the radio on?	NA	boolean
rolePlatform	Platform role	What role does the platform play, e.g., commander, leader, relay?	NA	String
type	Type	Type of platform, e.g., jammer, platform description.	NA	String
urn	Uniform resource number	Unique identification used in the U.S. Army, for each platform, similar to serial number.	NA	String

RadioModelRed

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
bandwidth	Bandwidth	Bandwidth of radiowaveform.	MHz	double
bw60db	Bandwidth 60 dB down	In selectivity, stopband at (-60dB).	MHz	double
bw6db	Bandwidth 6 dB down	In selectivity, passband at (-6dB).	MHz	double
frequency	Frequency	Frequency of radio.	MHz	double
insertionLoss	Insertion loss	Insertion loss in selectivity.	Decibels	double
noiseEquivalentBandwidth	Noise equivalent bandwidth	Noise equivalent bandwidth of radio.	Megahertz	double
noiseFigure	Noise figure	Noise figure for radio.	Decibels	double
power	Power	Radio power.	Watts	double
radio	Radio	Radio name as seen by user in deployment setup; radios tab “Type” combo selection. Values are shown from red database, e.g., SINCGARS, EPLRS.	NA	String
sensitivity	Sensitivity	Sensitivity of radio.	Decibel milliseconds	double

Slice

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
primaryKey	Primary key	Captures primary key for each record/row in table.	NA	Long
azimuth	Azimuth	Azimuth angle of slice for a 3-D antenna pattern. The azimuth angle progresses counterclockwise starting at north = 0 rad, i.e., math notation.	Radians	double
maxAngle	Maximum angle	Maximum angle that slices are available for the 3-D antenna pattern; usually 6.28 rad, i.e., 2 Pi.	Radians	double
minAngle	Minimum angle	Minimum angle that slices are available for the 3-D antenna pattern; usually 0 rad.	Radians	double
AntennaPattern_id	Antenna pattern ID number	Foreign key for linking this table to the antenna pattern table.	NA	Long

Slice_angle

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
Slice_primaryKey	Foreign key linking this entry to the slice	Captures foreign key for each row linking this entry to the slice table; since Hibernate generates these keys from an array persistence model, no primary keys are generated; only foreign keys.	NA	Long
angle	Elevation angle within the slice	Elevation angle starting at 0 rad or horizontal to the tangent to the earth's surface, and progressing upward.	Radians	double
angle_index	Angle index	Hibernate generates an index for each elevation entry row. This corresponds to the same index as in the Slice_gain table. Thus, as the elevation progresses through each index, so do the corresponding values in the Slice_gain table. The Slice_primaryKey and the index link the values in this table to a unique slice.	NA	long

Slice_gain

Database Field Name/Java Variable	Descriptive Field Name	Description or Purpose	Units	Type
Slice_primaryKey	Foreign key linking this entry to the slice	Captures foreign key for each for linking this entry to the slice table; since Hibernate generates these keys from array persistence models no primary keys are generated; only foreign keys.	NA	Long
gain	Gain	Gain from antenna pattern at the slice, and elevation angle identified by the gain_index and Slice_prmaryKey.	Decibels	double
gain_index	Gain index	Hibernate generates an index for each gain row entry. This corresponds to the same index as in the Slice_angle table. The Slice_primaryKey and the gain index link the values in this table to a unique slice.	NA	long

INTENTIONALLY LEFT BLANK.

Appendix B. Setting Up the Database Development Environment

Getting Started

To develop an application that needs to save Java objects to a database using Hibernate, the following application development environment is required:

- The open source MySQL Database Management System (DBMS) from Oracle, which is a Java Database Connectivity (JDBC)–compliant DBMS
- MySQL JDBC Driver library that comes as a plug-in with the Netbeans distribution
- The latest Java Development Kit with the latest Netbeans distribution, as of August 2012; version 7.2, which includes Java version 1.7
- The Hibernate application programming interface, which is a plug-in provided in Netbeans distribution, i.e., Hibernate Java persistence application programming interface, which includes the various jar files and libraries associated with Hibernate

Adding an Appropriate JDBC Driver

For a Java program to connect properly to a database it needs a JDBC driver. All major DBMS distributions come with a JDBC driver. Network Connectivity Analysis Model (NCAM) uses the MySQL DBMS, with a JDBC driver MySQL JDBC Driver. In the Netbeans integrated development environment, this is added to your project by right clicking on the project and selecting Properties-> add Library and navigating to where the file MySQL JDBC Driver is found. If it is not there, you need to download from the Netbeans Tools-> Plugins and select the Available Plugins tab to download.

Setting Up and Administering MySQL

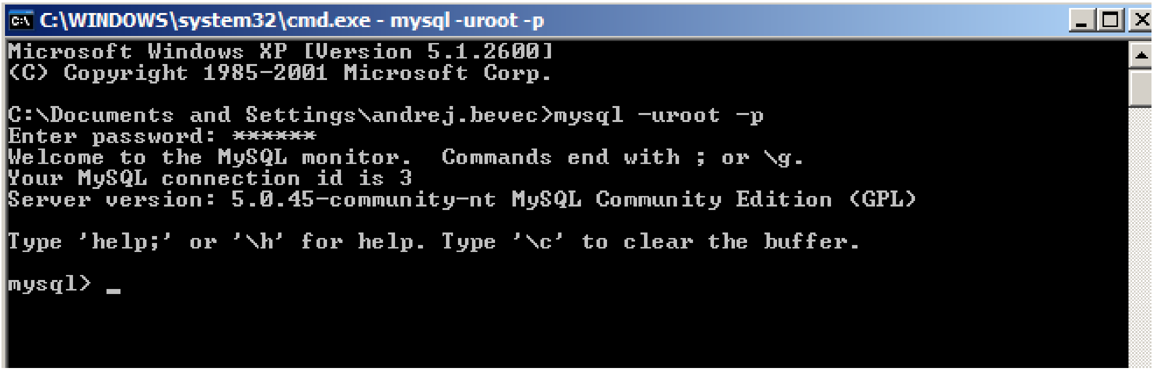
Download the MySQL distribution from the MySQL website[*] and install it per documentation onto your database server machine or onto your local machine depending on your work. Also install the graphic user interface (GUI) administrator tools provided for MySQL. The latest version is MySQL Workbench or MySQL Query Brower, Administrator, etc.

Once MySQL is installed, you need to create a user account for the application that will connect to MySQL.

- Start up the MySQL console via the command line window via the GUI or in a Windows command line window, or a terminal window in Linux with

  mysql –uroot –p hit return,

[*] MySQL Home Page. http://dev.mysql.com/downloads/mysql/ (accessed 2 March 2010).



This signs you in as root user to the command console, which is set up during the MySQL DBMS installation onto the server or local machine.

- Create a database for the NCAM application; in our case we call it the “NCAM” database.

  mysql > create database NCAM;

- Add new accounts and grant proper privileges to the user, i.e., , ‘cmuser’ with all privileges to MySQL

  To grant all privileges to cmuser connecting from anywhere, ie, ‘%’
  *mysql> grant all privileges on *.* to ‘cmuse’r@’%’ identified by ‘cmpasswd’;*

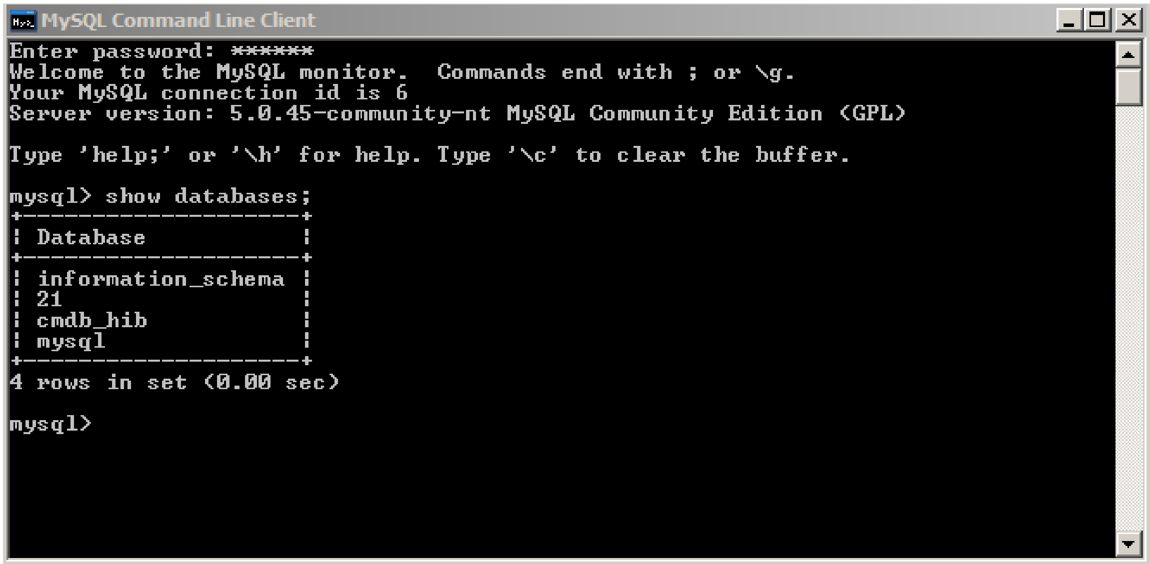
  Grant all privileges to cmuser connecting from local host with grant option
  *mysql > Grant all Privileges on *.* to ‘cmuser’@’localhost’ identified by ‘cmpasswd’ with grant option;*

  Grant all privileges to cmuser connecting from 128.63.62.16
  *mysql> grant all privileges on *.* ‘cmuser’@’128.63.62.16 identified by ‘cmpasswd’ with grant option;*

The account has a username of *cmuser* and a password of *cmpasswd*. It is a super user account with full privileges to do anything. As a database administrator you would want to limit these restrictions.

- Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
```



```
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.45-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| 21                 |
| cmdb_hib           |
| mysql              |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

Other useful MySQL commands via the console are:

- If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create the database itself:

```
mysql> CREATE DATABASE NCAM;
```

- Creating Usernames and passwords

When you connect to a MySQL server with a command-line client, you should specify the username and password for the account that you want to use:

```
C:\> mysql --user=cmuser --password=cmpasswd NCAM
```

If you prefer short options, the command looks like this: `C:\> mysql -u cmuser -pcmpasswd NCAM`

There must be *no space* between the `-p` option and the following password value.

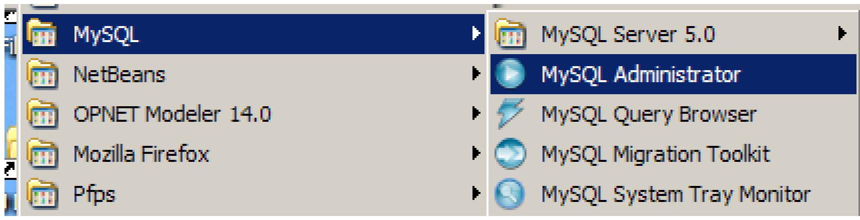
- Creating a database does not select it for use; you must do that explicitly. To make `NCAM` the current database, use this command:

```
mysql> USE NCAM;
Database changed
```

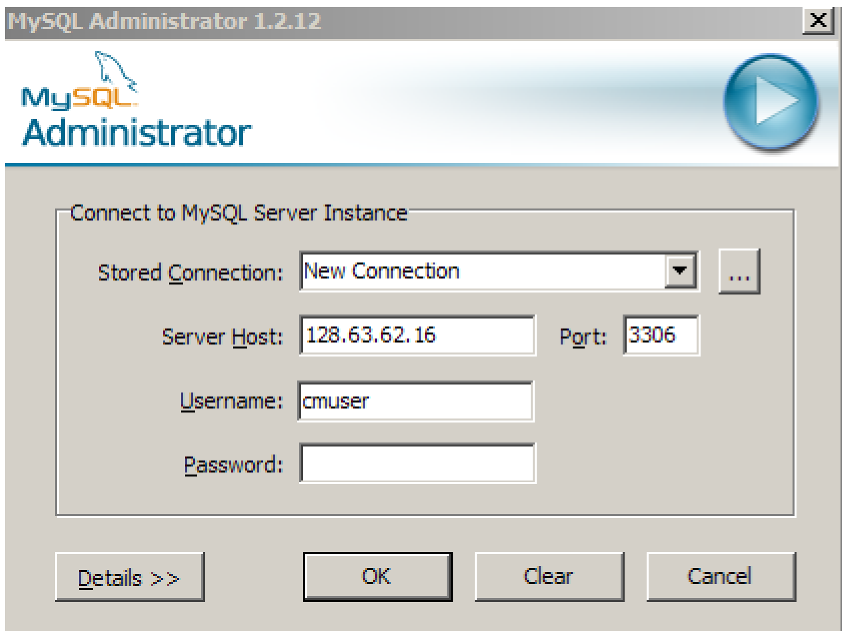
- Creating the database is the easy part, but at this point it's empty, as `SHOW TABLES` shows for the database you have previously select to use:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

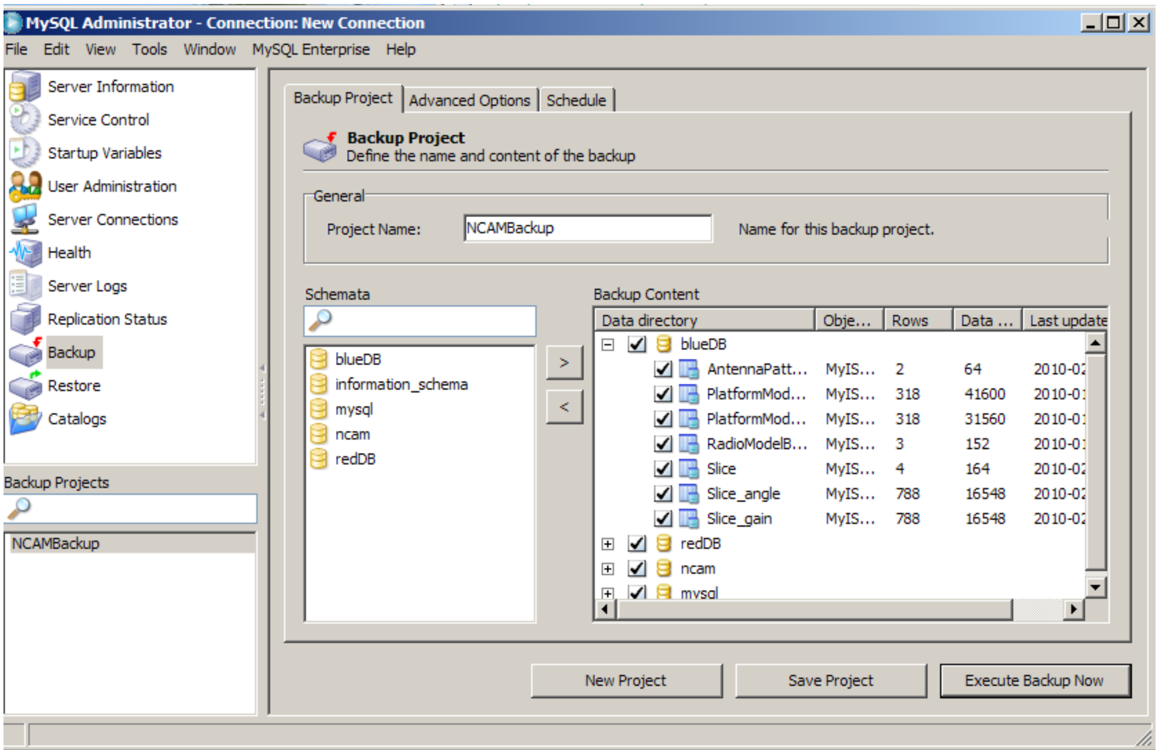
- Creating a backup with MySQL Administrator, go into the MySQL and start MySQL Administrator:



Sign in as 'cmuser' with password 'passwd'.

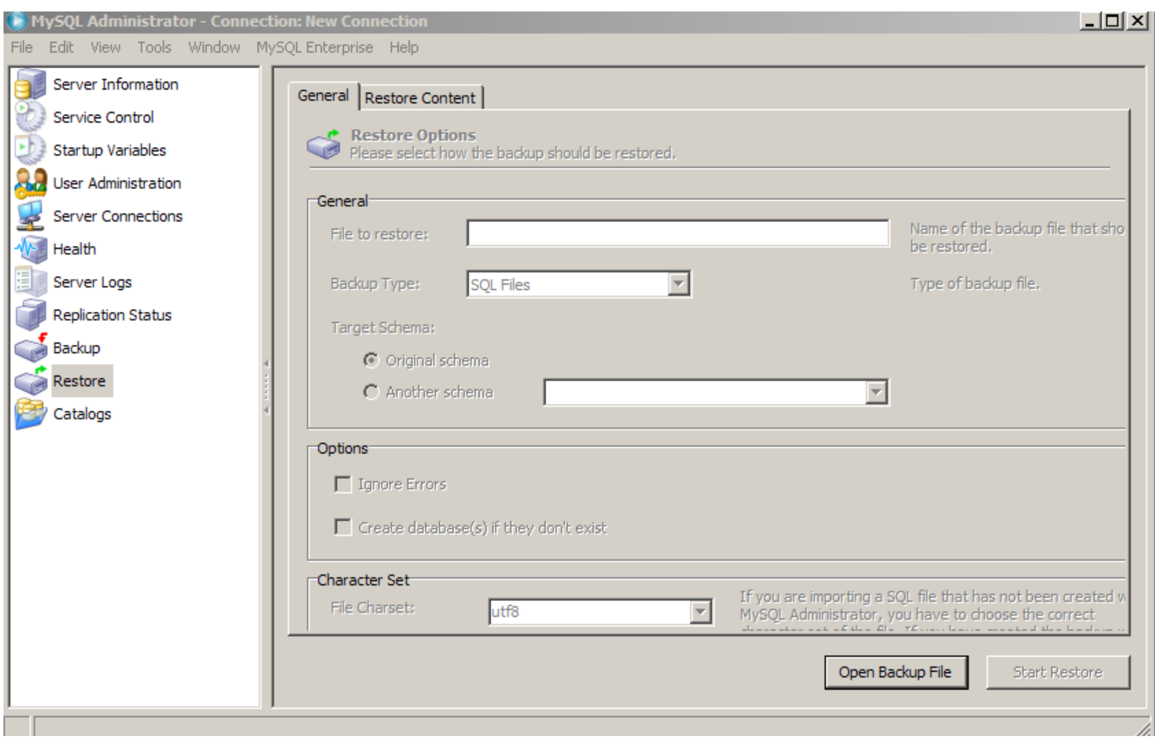


When the Administrator GUI appears, select Backup icon from the left panel, select the databases that you wish to backup, give it a project name, and click on Execute Backup Now button. The backup is stored as an SQL file.



- Restoring Backed up database files via the MySQL Administrator GUI

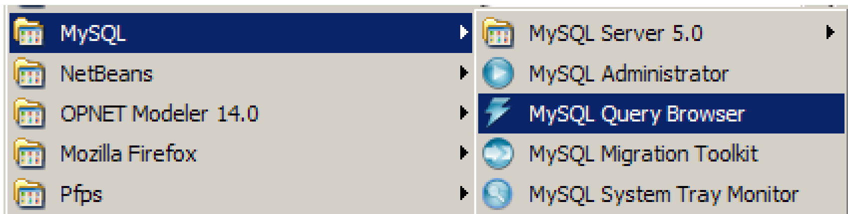
  Select the restore icon and navigate to your backup file by pressing the Open Backup File button.



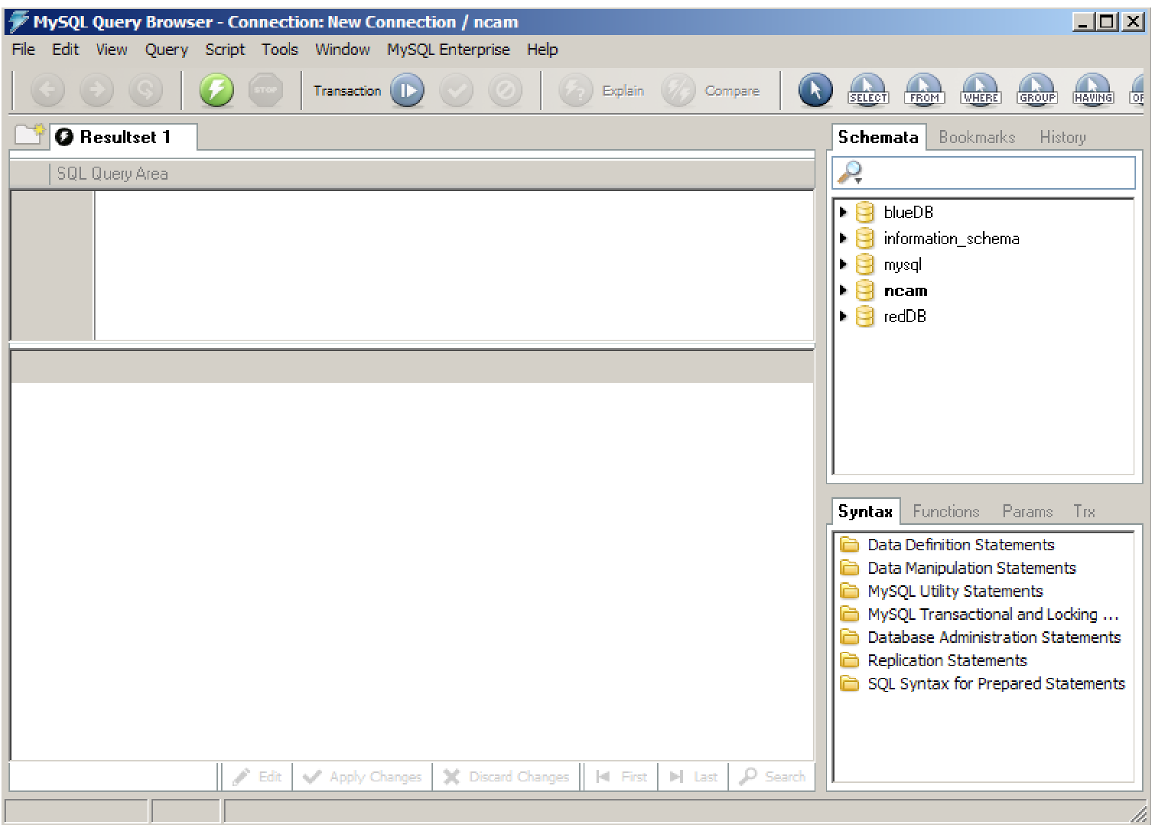
Using MySQL Query Browser

If you do not wish to use the MySQL command console, the GUI MySQL Query Browser provides a very intuitive interface for completing database management functions, e.g., creating new databases, creating a new table, dropping database schemas, and manually inserting database values. The MySQL Query Browser has been replaced by Oracle with the MySQL Workbench, with similar or better capabilities. This project used MySQL Query Browser on the Linux platform.

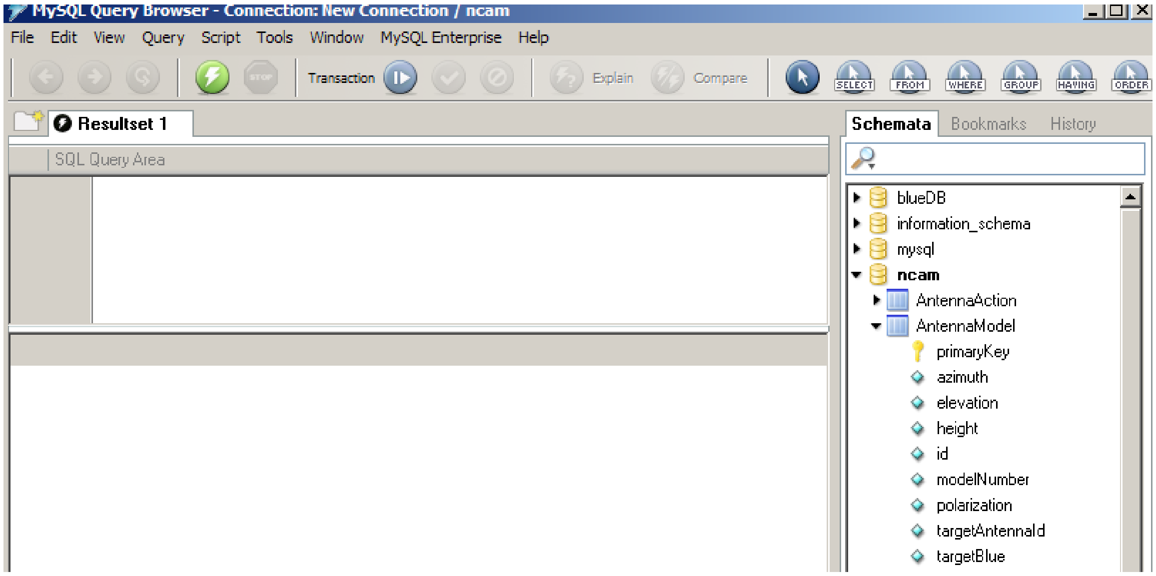
Select the MySQL Query Browser from the Start->Programs Windows menu.



In the Query Browser Schemata Panel, double click the database to view the tables.



Expanding the tick mark next to the database reveals the tables; expanding the tables tick mark exposes the database values.



Double-clicking on a table automatically generates the SQL query to print out all table results into the preview window. The SQL Query Area window may be used to write your own SQL queries as you would in a MySQL console window.

Bibliography

Bauer, C.; King, G. Java Persistence with Hibernate. In *Hibernate in Action*, 2nd Ed.; Manning Publications Co.: Shelter Island, NY, 2007.

Elliot, J.; O'Brien, T.; Fowler, R. *Harnessing Hibernate;* O'Reilly Media: Sebastopol, CA, 2008.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software;* Addison-Wesley Professional: Boston, MA, 1994.

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (HC)	DIR US ARMY EVALUATION CTR HQ TEAE SV P A THOMPSON 2202 ABERDEEN BLVD 2ND FL APG MD 21005-5001
5 (2 HC 3 PDF)	DIR USARL RDRL SL J BEILFUSS (HC) P TANENBAUM (HC) RDRL SLB S M PERRY RDRL SLE R FLORES RDRL SLE W A BEVEC